SDK-86
MCS-86™ SYSTEM DESIGN KIT
MONITOR LISTINGS

Manual Order Number: 9800699-03 Rev. C

Intel Corporation, 3065 Bowers Avenue, Santa Clara, California 95051

Additional copies of this manual or other Intel literature may be obtained from:

Literature Department
Intel Corporation
3065 Bowers Avenue
Santa Clara, CA 95051

The information in this document is subject to change without notice.

Intel Corporation makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Intel Corporation assumes no responsibility for any errors that may appear in this document. Intel Corporation makes no commitment to update nor to keep current the information contained in this document.

Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied.

The following are trademarks of Intel Corporation and may be used only to identify Intel products:

i	iSBC	Multimodule
ICE	Library Manager	PROMPT
iCS	MCS	Promware
Insite	Megachassis	RMX
Intel	Micromap	UPI
Intelevision	Multibus	μScope
Intellec		

and the combination of ICE, iCS, iSBC, MCS, or RMX and a numerical suffix.

PREFACE

This manual contains the source listing for version 1.1 of the keypad monitor program and version 1.2 of the serial monitor program. Two listings are provided for each monitor program; a PL/M-86 source listing and an expanded source listing containing the assembly language equivalent of each PL/M instruction.

SDK-86 KEYPAD MONITOR
PL/M-86 SOURCE LISTING


```
ISIS-II PL/M-86 V1.0 COMPILATION OF MODULE MONITOR
OBJECT MODULE PLACED IN :F1:KEYPAD.OBJ
COMPILER INVOKED BY:  :F2:PLM86 :F1:KEYPAD.PLM LARGE OPTIMIZE(2)


$TITLE('SDK86 KEYPAD MONITOR')
$NOINTVECTOR

 /* ***********************************************************************
    ***********************************************************************

        SDK-86 KEYPAD MONITOR, V1.1
        30 JUNE 1978

      (C) 1978 INTEL CORPORATION. ALL RIGHTS RESERVED. NO PART OF
    OF THIS PROGRAM OR PUBLICATION MAY BE REPRODUCED, TRANSMITTED,
    TRANSCRIBED, STORED IN A RETRIEVAL SYSTEM, OR TRANSLATED INTO ANY
    LANGUAGE, IN ANY FORM OR BY ANY MEANS, ELECTRONIC, MECHANICAL,
    MAGNETIC, OPTICAL, CHEMICAL, MANUAL OR OTHERWISE, WITHOUT THE PRIOR
    WRITTEN PERMISSION OF INTEL CORPORATION, 3065 BOWERS AVENUE, SANTA
    CLARA, CALIFORNIA, 95051.

    ***********************************************************************
    ***********************************************************************

    ABSTRACT
    ========
    THIS PROGRAM IS THE ROM BASED KEYPAD MONITOR FOR THE SDK-86.
    IT PROVIDES THE USER WITH A MODERATE LEVEL OF CAPABLILITY TO
    EXAMINIE/MODIFY MEMORY AND REGISTERS AND CONTROL PROGRAM EXECUTION.

    ENVIRONMENT
    ===========
    THE SDK KEYPAD MONITOR COMMUNICATES WITH THE USER VIA THE ONBOARD
    KEYBOARD AND 7 SEGMENT DISPLAY.

    PROGRAM ORGANIZATION
    ====================
    THE PROGRAM IS DIVIDED INTO 1 DATA AND 2 CODE MODULES:
       1. DATA DECLARATION MODULE.     GLOBAL DATA DECLARATIONS.
       2. COMMON ROUTINES.             LOWER LEVEL PROCEDURES
       3. COMMAND MODULE.              INDIVIDUAL COMMANDS AND OUTER BLOCK

    CALLING GRAPH
    =============
       >>COMMAND DISPATCH MODULE (OUTER BLOCK)
            INDIVIDUAL COMMAND PROCEDURES
               COMMON ROUTINES

    GLOBAL DATA STRUCTURES
    ======================
    THE MONITOR MAINTAINS THE USER'S MACHINE STATE (REGISTERS) IN A
    WORD ARRAY. THE REGISTERS ARE SAVED FROM THE USER'S STACK
    AS PUSHED BY PLM86 INTERRUPT PROCEDURE.
    POINTERS TO THE SDK-86 2**20 ADDRESS SPACE ARE IMPLEMENTED WITH
    POINTER STRUCTURES ALLOCATED AS 2 WORD STRUCTURES.
```

```
                */

  1             MONITOR:DO;                     /* BEGINNING OF MODULE */

                /* ***************************************************************
                   ***************************************************************

                                  GLOBAL DATA DECLARATIONS MODULE
                                  ===============================
                   ABSTRACT
                   ========
                   THIS MODULE CONTAINS ALL THE GLOBAL DATA DECLARATIONS AND
                   LITERALS (EQUATES).

                   MODULE ORGAINIZATION
                   ====================
                   THE MODULE IS DIVIDED INTO 5 SECTIONS:
                      1. UTILITY SECTION              GLOBAL FLAGS,VARIBLES,EQUATES
                      2. I/O SECTION                  I/O PORTS,MASKS,AND SPECIAL CHARS
                      3. MEMORY ARGUMENTS SECTIONS    STRUCTURES FOR MEDIUM POINTERS
                      4. REGISTER SECTION             USER REGISTER SAVE AREA
                      5. BOOT AND 8089 SECTION        BOOTSTRAP AND 8089 DESCRIPTOR
                */

                  /***********************
                   *   UTILITY SECTION   *
                   ***********************/

  2   1         DECLARE
                      INT$VECTOR(5)    POINTER;              /* INTERRUPT VECTORS */
  3   1         DECLARE
                      MONITOR$STACKPTR  WORD,
                      MONITOR$STACKBASE WORD;                /* MONITOR SS SAVE */

  4   1         DECLARE
                      COPYRIGHT(*) BYTE DATA ('(C) 1978 INTEL CORP');
  5   1         DECLARE
                      BRK1$FLAG       BYTE,                  /* TRUE IF BREAK SET */
                      BRK1$SAVE       BYTE,                  /* INST BREAK SAVE */
                      CHAR            BYTE,                  /* ONE CHAR LOOK AHEAD */
                      DISP(4)         BYTE,                  /* DISPLAY ARRAY  */
                      I               BYTE,
                      END$OFF         WORD,                    /* END OFFSET ADDRESS */
                      WORD$MODE       BYTE,
                      LAST$COMMAND    BYTE;
  6   1         DECLARE
                      TRUE            LITERALLY '0FFH',
                      FALSE           LITERALLY '000H',
                      ADDR$FIELD      LITERALLY '1',         /* ADDR FIELD WORD OUTPUT */
                      DATA$FIELD      LITERALLY '0',         /* DATA FIELD WORD OUTPUT */
                      DATA$BYTE       LITERALLY '-1',        /* DATA FIELD BYTE OUTPUT */
                      BLANK           LITERALLY '1',
                      NOBLANK         LITERALLY '0',
                      BREAK$INST      LITERALLY '0CCH',      /* BREAKPOINT TRAP */
                      STEP$TRAP       LITERALLY '0100H',     /* SS TRAP FLAG MASK */
                      USER$INIT$SP    LITERALLY '100H',      /* USER STACK INITIAL */
```

```
                GO$COMMAND      LITERALLY '2',          /* GO COMMAND CODE */
                SS$COMMAND      LITERALLY '3';

              /*******************************
              *   I/O DECLARATIONS SECTION   *
              *******************************/

  7   1       DECLARE
                KB$STAT$PORT    LITERALLY 'OFFEAH',           /* STATUS/COMMAND PORT */
                KB$DATA$PORT    LITERALLY 'OFFE8H',           /* DATA PORT */
                KB$INIT$MODE    LITERALLY '00H',              /* 8 8-BIT, LEFT ENTRY,
                                                                 ENCODE, 2 KEY LOCKOUT */
                KB$INIT$SCAN    LITERALLY '39H',              /* 10MS SCAN RATE   */
                KB$INRDY        LITERALLY '07H',              /* IN CHAR RDY MASK */
                KB$CMND$PMT(*)  BYTE DATA (40H,00H,00H,00H), /* '-   ' */
                KB$SIGNON(*)    BYTE DATA (40H,00H,7FH,7DH), /* '- 86' */
                KB$VERSION(*)   BYTE DATA (00H,00H,86H,06H), /* '  1.1' */
                KB$REG$PMT(*)   BYTE DATA (50H,00H,00H,00H), /* 'R   ' */
                KB$ERR$MSG(*)   BYTE DATA (40H,79H,50H,50H), /* '-ERR' */
                KB$EXEC(*)      BYTE DATA (79H,00H,00H,00H), /* 'E   ' */
                KB$BRK1(*)      BYTE DATA (7CH,50H,00H,00H), /* 'BR  ' */
                KB$BLANKS(*)    BYTE DATA (00H,00H,00H,00H), /* '    ' */
                LED(*)          BYTE DATA
                   (3FH,06H,5BH,4FH,66H,6DH,7DH,07H,          /* '0'-'7' */
                    7FH,6FH,77H,7CH,39H,5EH,79H,71H);         /* '8'-'F' */
  8   1       DECLARE
                KBPER           LITERALLY  '10H',             /* PERIOD          */
                KBCOM           LITERALLY  '11H',             /* COMMA           */
                KBREGKEY        LITERALLY  '15H',             /* REGISTER KEY    */
                KBCOL           LITERALLY  '14H',             /* COLON (SEGMENT) */
                KBPLUS          LITERALLY  '13H',             /* PLUS KEY        */
                KBMINUS         LITERALLY  '12H';             /* MINUS KEY       */

              /***********************
              *   POINTER SECTION   *
              ***********************/

  9   1       DECLARE
                MEMORY$ARG1$PTR  POINTER,               /* ARGUMENT 1 */
                ARG1 STRUCTURE (OFF WORD, SEG WORD)
                   AT (@MEMORY$ARG1$PTR),
                MEMORY$ARG1 BASED MEMORY$ARG1$PTR BYTE,
                MEMORY$WORD$ARG1 BASED MEMORY$ARG1$PTR WORD,

                MEMORY$ARG3$PTR POINTER,                /* ARGUMENT 3 */
                ARG3 STRUCTURE (OFF WORD, SEG WORD)
                   AT (@MEMORY$ARG3$PTR),
                MEMORY$ARG3 BASED MEMORY$ARG3$PTR BYTE,

                MEMORY$BRK1$PTR POINTER,                /* BREAKPOINT    */
                BRK1 STRUCTURE (OFF WORD, SEG WORD)
                   AT (@MEMORY$BRK1$PTR),
                MEMORY$BRK1 BASED MEMORY$BRK1$PTR BYTE,

                MEMORY$CSIP$PTR POINTER,                /* CS:IP INSTRUCTION */
                CSIP STRUCTURE (OFF WORD, SEG WORD)
                   AT (@MEMORY$CSIP$PTR),
```

```
        MEMORY$CSIP BASED MEMORY$CSIP$PTR BYTE,

        MEMORY$USERSTACK$PTR POINTER,           /* USER'S STACK  */
        USERSTACK STRUCTURE (OFF WORD, SEG WORD)
           AT (@MEMORY$USERSTACK$PTR),
        MEMORY$USERSTACK BASED MEMORY$USERSTACK$PTR WORD;


     /************************
     *   REGISTER SECTION   *
     ************************/

10   1  DECLARE
           KB$REG(*)       BYTE DATA
              (00H,77H,00H,7CH,00H,39H,00H,5EH,       /* AXBXCXDX   */
               6DH,73H,7CH,73H,6DH,30H,5EH,30H,       /* SPBPSIDI   */
               39H,6DH,5EH,6DH,6DH,6DH,79H,6DH,       /* CSDSSSES   */
               30H,73H,71H,38H),                      /* IPFL       */
           REG$SAV(14)     WORD,                /* USER'S SAVED REGS */
           REG$ORD(*)      BYTE DATA
              (7,6,1,3,2,0,9,11,12,8,13),      /* STACKED REG ORDER */
           SP              LITERALLY 'REG$SAV( 4)',
           BP              LITERALLY 'REG$SAV( 5)',
           CS              LITERALLY 'REG$SAV( 8)',
           DS              LITERALLY 'REG$SAV( 9)',
           SS              LITERALLY 'REG$SAV(10)',
           ES              LITERALLY 'REG$SAV(11)',
           IP              LITERALLY 'REG$SAV(12)',
           FL              LITERALLY 'REG$SAV(13)';


     /************************************
     *   BOOTSTRAP JUMP AND 8089 VECTOR   *
     ************************************/

     /* THIS BOOT CONSISTS OF A LONG JUMP TO THE BEGINNING OF THE MONITOR
        AT FF00+XXXX WHERE XXXX IS THE STARTING ADDRESS OFFSET (IP) AND
        MUST BE DETERMINED AFTER EACH COMPILE. */
11   1  DECLARE
           START$ADDR LITERALLY '009CH',               /* STARTING ADDRESS */
           BOOT1(*)   BYTE AT (0FFFF0H) DATA (0EAH),   /* LONG JUMP OPCODE */
           BOOT2(*)   WORD AT (0FFFF1H) DATA (START$ADDR),
           BOOT3(*)   WORD AT (0FFFF3H) DATA (0FF00H); /* SEGMENT ADDRESS  */

     /* THIS TWO-WORD DATA IS A JUMP TO THE STARTING ADDRESS AND IS LOCATED
        AT THE FIRST LOCATION OF ROM (NO OTHER DATA OR CONSTANT DECLARATIONS
        MAY PRECEDE IT). THE JUMP IS ACTUALLY TO (START-ADDR)-4 SINCE THE
        INSTRUCTION IS A RELATIVE JUMP OF LENGTH 3. */
12   1  DECLARE
           BOOT4(*)   WORD DATA (0E990H,START$ADDR-4);  /* NOP,JMP START-ADDR */

     /* THIS BLOCK OF ROM AT FFFF6-FFFFA IS INITIALIZED FOR THE 8089
         DEVICE AND POINTS TO A BLOCK OF RAM AT LOCATION 100H. */
13   1  DECLARE
           BLOCK$8089       WORD    AT (0FFFF6H) DATA (00001H),
           BLOCK$8089$PTR   POINTER AT (0FFFF8H) DATA (00100H);
```

```
/* ************************************************************************
 ************************************************************************

                    COMMON PROCEDURES
                    =================
   ABSTRACT
   ========
   THIS MODULE CONTAINS THOSE LOWER LEVEL PROCEDURES CALLED BY HIGHER
   LEVEL ROUTINES.

   MODULE ORGANIZATION
   ===================
   THIS MODULE CONTAINS THE FOLLOWING SECTIONS:
      1. BASIC I/O SECTION
          KB$DISPLAY              DISPLAY TO LED FIELDS
          KB$BLANK$DATA$FIELD     BLANK DATA FIELD WITH PROMPTS
          KB$BLANK$ADDR$FIELD     BLANK ADDRESS FIELD WITH PROMPTS
          KB$OUT$BYTE             OUTPUT A BYTE TO DISPLAY
          KB$OUT$WORD             OUTPUT A WORD TO DISPLAY
          KB$GET$CHAR             INPUT A CHAR FROM KEYPAD
      2. ARGUMENT EXPRESSION EVALUATOR
          KB$GET$EXPR             GET WORD EXPRESSION
          KB$GET$ADDR             GET ADDRESS EXPRESSION
          KB$UPDATE$IP            GET OPTIONAL CS:IP
      3. INTERRRUPT AND RESTORE/EXECUTE ROUTINES
          SAVE$REGISTERS          SAVES USERS REGISTERS
          RESTORE$EXECUTE         RESTORE MACHINE STATE AND EXEC
          INTERRUPR1$ENTRY        INTERRUPT FOR SINGLE STEP
          INITERRUPT3$ENTRY       INTERRUPT ROUTINE FOR GO
          INIT$INT$VECTOR         INITIALIZES INTERRRUPT VECTORS
*/

   /************************
   *   BASIC I/O SECTION   *
   ************************/

14  1    KB$DISPLAY:
           /* THIS ROUTINE DISPLAYS THE CONTENTS OF THE ARRAY POINTED TO BY
              THE FIRST PARM TO THE FIELD SPECIFIED BY THE SECOND (ADDR OR
              DATA). THE NUMBER OF DECIMAL POINTS OR PROMPTS IS DETERMINED
              BY THE THIRD PARAMETER */
           PROCEDURE (PTR,FIELD,PROMPTS);
15  2          DECLARE PTR POINTER, (FIELD,PROMPTS,T) BYTE,
                  DISPLAY BASED PTR (1) BYTE;
16  2          IF FIELD=ADDR$FIELD THEN
17  2             OUTPUT(KB$STAT$PORT) = 94H;          /* ADDRESS FIELD */
               ELSE
18  2             OUTPUT(KB$STAT$PORT) = 90H;          /* DATA FIELD */
19  2          DO I=0 TO 3;
20  3             T = DISPLAY(3-I);                       /* DISPLAY BACKWARDS! */
21  3             IF PROMPTS>I THEN  T = T OR 80H;
23  3             OUTPUT(KB$DATA$PORT) = T;
24  3          END;
25  2        END;

26  1    KB$BLANK$DATA$FIELD:
```

```
                     /* THIS ROUTINE BLANKS THE DATA FIELD OF THE DISPLAY WITH THE
                        NUMBER OF PROMPTS (DECIMAL POINTS) AS SPECIFIED BY THE PARAMETER. */
                     PROCEDURE(PROMPTS);
  27    2                DECLARE PROMPTS BYTE;
  28    2                CALL KB$DISPLAY(@KB$BLANKS,DATA$FIELD,PROMPTS);
  29    2             END;

  30    1          KB$BLANK$ADDR$FIELD:
                     /* THIS PROCEDURE BLANKS THE ADDRESS FIELD OF THE DISPALY WITH THE
                        NUMBER OF PROMPTS SPECIFIED BY THE PARAMETER 'PROMPTS'. */
                     PROCEDURE(PROMPTS);
  31    2                DECLARE PROMPTS BYTE;
  32    2                CALL KB$DISPLAY(@KB$BLANKS,ADDR$FIELD,PROMPTS);
  33    2             END;

  34    1          KB$OUT$BYTE:
                     /* THIS ROUTINE OUTPUTS THE BYTE INPUT FROM THE FIRST PARAMETER
                        TO THE DATA FIELD WITH THE NUMBER OF PROMPTS SPECIFIED BY THE
                        SECOND PARAMETER. */
                     PROCEDURE(B,PROMPTS);
  35    2                DECLARE (B,PROMPTS) BYTE;
  36    2                DISP(0),DISP(1) = 0;                    /* FIRST TWO BLANK */
  37    2                DISP(2) = LED(SHR(B,4) AND 0FH);
  38    2                DISP(3) = LED(B AND 0FH);
  39    2                CALL KB$DISPLAY(@DISP,DATA$FIELD,PROMPTS);
  40    2             END;

  41    1          KB$OUT$WORD:
                     /* THIS ROUTINE OUTPUTS THE FIRST PARM TO THE FIELD SPECIFIED
                        BY THE SECOND WITH THE NUMBER OF PROMPTS SPECIFIED BY THE THIRD.
                        LEADING ZERO BLANKING IS PERFORMED IF SPECIFIED BY THE FOURTH
                        PARAMETER. */
                     PROCEDURE(W,FIELD,PROMPTS,BLANKING);
  42    2                DECLARE W WORD, (FIELD,PROMPTS,BLANKING) BYTE;
  43    2                DO I=0 TO 3;
  44    3                   DISP(I) = LED(SHR(W,(3-I)*4) AND 000FH);
  45    3                END;
  46    2                IF BLANKING=BLANK THEN                  /* BLANK LEADING 0'S */
  47    2                   DO;
  48    3                   I = 0;
  49    3                   DO WHILE DISP(I)=3FH AND I<3;
  50    4                      DISP(I) = 0;
  51    4                      I = I + 1;
  52    4                   END;
  53    3                   END;
  54    2                CALL KB$DISPLAY(@DISP,FIELD,PROMPTS);
  55    2             END;

  56    1          KB$GET$CHAR:
                     /* READS ONE CHARACTER FROM THE FIFO OF THE 8279. WAITS UNTIL
                        CHARACTER IS AVAILABLE AND THEN RETURNS THE CHARACTER IN GLOBAL
                        VARIABLE 'CHAR'. */
                     PROCEDURE;
  57    2                DO WHILE (INPUT(KB$STAT$PORT) AND KB$INRDY)=0; END;
  59    2                OUTPUT(KB$STAT$PORT) = 040H;            /* ENABLE INPUT DATA */
  60    2                CHAR = INPUT(KB$DATA$PORT);             /* READ CHARACTER */
  61    2             END;
```

```
                /*********************************************
                *   ARGUMENT EXPRESSION EVALUATOR SECTION    *
                *********************************************/

 62   1       KB$GET$EXPR:
                /* THIS ROUTINE GATHERS CHARACTERS FROM THE INPUT STREAM
                   AND FORMS A WORD EXPRESSION WHICH IS RETURNED AS THE VALUE
                   OF THE PROCEDURE. THE CHARACTERS INPUT ARE ECHOED TO THE
                   ADDRESS OR DATA FIELD AS SPECIFIED BY THE FIRST PARAMETER.
                   IF THE FIRST PARM IS 0 (ADDR) OR 1 (DATA) THEN THE EXPRESSION IS
                   OUTPUT AS A WORD AND IF -1 (DATA$BYTE) THEN AS A BYTE ALWAYS IN
                   DATA FIELD. THE NUMBER OF PROMPTS BY THE SECOND PARM, AND LEADING
                   ZERO BLANKING BY THE THIRD.  EXPRESSION ARE TERMINATED WITH
                   A COMMA,PERIOD OR COLON. 'CHAR' WILL CONTAIN ONE OF THESE
                   ON EXIT. */
                PROCEDURE (FIELD,PROMPTS,BLANKING) WORD;
 63   2             DECLARE (FIELD,PROMPTS,BLANKING) BYTE,
                      (SAVE,W) WORD, OPER BYTE;
 64   2             OPER = KBPLUS;
 65   2             W = 0;
 66   2             DO WHILE TRUE;
 67   3                IF CHAR=KBREGKEY THEN
 68   3                   DO;                                    /* REGISTER NAME */
 69   4                   CALL KB$DISPLAY(@KB$REG$PMT,FIELD,PROMPTS);
 70   4                   CALL KB$GET$CHAR;
 71   4                   IF CHAR>0DH THEN  GOTO ERROR;          /* INVALID REG KEY */
 73   4                   SAVE = REG$SAV(CHAR);
 74   4                   IF FIELD=DATA$BYTE THEN
 75   4                      CALL KB$OUT$BYTE(LOW(SAVE),PROMPTS);
                           ELSE
 76   4                      CALL KB$OUT$WORD(SAVE,FIELD,PROMPTS,BLANKING);
 77   4                   CALL KB$GET$CHAR;
 78   4                   END;
                      ELSE
 79   3                   DO;                                    /* NUMBER */
 80   4                   IF CHAR>0FH THEN  GOTO ERROR;          /* INVALID DIGIT */
 82   4                   SAVE = 0;
 83   4                   DO WHILE CHAR<=0FH;
 84   5                      SAVE = SHL(SAVE,4) + DOUBLE(CHAR);
 85   5                      IF FIELD=DATA$BYTE THEN
 86   5                         CALL KB$OUT$BYTE(LOW(SAVE),PROMPTS);
                              ELSE
 87   5                         CALL KB$OUT$WORD(SAVE,FIELD,PROMPTS,BLANKING);
 88   5                      CALL KB$GET$CHAR;
 89   5                   END;
 90   4                   END;
 91   3                IF OPER=KBPLUS THEN                       /* EVAL PREV OPER */
 92   3                   W = W + SAVE;
                      ELSE
 93   3                   W = W - SAVE;
 94   3                IF FIELD=DATA$BYTE THEN
 95   3                   CALL KB$OUT$BYTE(LOW(W),PROMPTS);
                      ELSE
 96   3                   CALL KB$OUT$WORD(W,FIELD,PROMPTS,BLANKING);
 97   3                IF CHAR=KBCOM OR CHAR=KBPER OR CHAR=KBCOL THEN
```

```
 98   3                  RETURN W;
 99   3               IF CHAR=KBPLUS OR CHAR=KBMINUS THEN
100   3                  OPER = CHAR;
                      ELSE
101   3                  GOTO ERROR;
102   3               CALL KB$GET$CHAR;                    /* GET NEXT CHAR */
103   3            END;
104   2         END;

105   1      KB$GET$ADDR:
            /* THIS ROUTINE GATHERS CHARACTERS FROM THE INPUT STREAM AND
               FORMS AN ADDRESS EXPRESSION OF SEGMENT PART AND OFFSET PART.
               THE FIRST CHARACTER HAS ALREADY BEEN READ INTO GLOBAL 'CHAR'.
               IF NO SEGMENT IS ENTERED, THE SECOMD PARM DEFAULT IS USED, THE
               ADDRESS EXPRESSION IS DISPLAYED IN THE ADDRESS FIELD WITH THE
               NUMBER OF PROMPTS SPECIFIED BY THE THIRD PARM. */
            PROCEDURE(PTR,DEFAULT$BASE,PROMPTS);
106   2        DECLARE PTR POINTER, DEFAULT$BASE WORD, PROMPTS BYTE,
                  ARG BASED PTR STRUCTURE (OFF WORD, SEG WORD);
107   2        ARG.SEG = DEFAULT$BASE;
108   2        ARG.OFF = KB$GET$EXPR(ADDR$FIELD,PROMPTS,BLANK);
109   2        IF CHAR=KBCOL THEN                       /* SEGMENT SPEC'D */
110   2           DO;
111   3           CALL KB$GET$CHAR;
112   3           ARG.SEG = ARG.OFF;
113   3           ARG.OFF = KB$GET$EXPR(ADDR$FIELD,PROMPTS,BLANK);
114   3           IF CHAR=KBCOL THEN  GOTO ERROR;
116   3           END;
117   2        END;

118   1      KB$UPDATE$IP:
            /* THIS ROUTINE IS CALLED BY SINGLE STEP AND GO TO OUTPUT THE CURRENT
               CS:IP AND THE CURRENT INSTRUCTION BYTE.  CS:IP IS OPENED FOR
               OPTIONAL INPUT. */
            PROCEDURE;
119   2        CALL KB$OUT$WORD(IP,ADDR$FIELD,1,BLANK);    /* DISPLAY IP */
120   2        CSIP.OFF = IP;
121   2        CSIP.SEG = CS;
122   2        CALL KB$OUT$BYTE(MEMORY$CSIP,0);
123   2        CALL KB$GET$CHAR;
124   2        IF CHAR<>KBCOM AND CHAR<>KBPER THEN
125   2           DO;                                   /* CHANGE CS:IP */
126   3           CALL KB$BLANK$ADDR$FIELD(1);
127   3           CALL KB$BLANK$DATA$FIELD(0);
128   3           CALL KB$GET$ADDR(@CSIP,CS,1);
129   3           END;
130   2        END;


            /*********************************************
            *   INTERRUPT AND RESTORE/EXECUTE SECTION   *
            *********************************************/

131   1      SAVE$REGISTERS:
            /* THIS ROUTINE IS USED TO SAVE THE STACKED USER'S REGISTERS IN THE
               MONITOR'S SAVE AREA. */
            PROCEDURE;
```

```
 132   2              BP = MEMORY$USERSTACK;
 133   2              USERSTACK.OFF = USERSTACK.OFF + 4;
 134   2              DO I=0 TO 10;                          /* POP REGISTERS OFF OF STACK */
 135   3                 REG$SAV(REG$ORD(I)) = MEMORY$USERSTACK;
 136   3                 USERSTACK.OFF = USERSTACK.OFF + 2;
 137   3              END;
 138   2              SS = USERSTACK.SEG;
 139   2              SP = USERSTACK.OFF;
 140   2           END;

 141   1        RESTORE$EXECUTE:
              /* THIS PROCEDURE RESTORES THE STATE OF THE USER MACHINE AND
                 PASSES CONTROL BACK TO THE USER PROGRAM. IT CONTAINS A
                 MACHINE LANGUAGE SUBROUTINE TO PERFORM THE POPPING OF THE
                 USER REGISTERS AND TO EXECUTE AN 'IRET' TO TRANSFER CONTROL
                 TO THE USER'S PROGRAM. */
              PROCEDURE;
 142   2              DECLARE RESTORE$EXECUTE$CODE(*) BYTE DATA
                       (08BH,0ECH,            /* MOV BP,SP            */
                        08BH,046H,002H,       /* MOV AX,/BP/.PARM2    */
                        08BH,05EH,004H,       /* MOV BX,/BP/.PARM1    */
                        08EH,0D0H,            /* MOV SS,AX            */
                        08BH,0E3H,            /* MOV SP,BX            */
                        05DH,                 /* POP BP               */
                        05FH,                 /* POP DI               */
                        05EH,                 /* POP SI               */
                        05BH,                 /* POP BX               */
                        05AH,                 /* POP DX               */
                        059H,                 /* POP CX               */
                        058H,                 /* POP AX               */
                        01FH,                 /* POP DS               */
                        007H,                 /* POP ES               */
                        0CFH),                /* IRET                 */
                       RESTORE$EXECUTE$CODE$PTR WORD DATA (.RESTORE$EXECUTE$CODE);

 143   2              USERSTACK.SEG = SS;
 144   2              USERSTACK.OFF = SP;
 145   2              DO I=0 TO 10;                  /* PUSH USER'S REGISTERS ONTO HIS STACK */
 146   3                 USERSTACK.OFF = USERSTACK.OFF - 2;
 147   3                 MEMORY$USERSTACK = REG$SAV(REG$ORD(10-I));
 148   3              END;
 149   2              USERSTACK.OFF = USERSTACK.OFF - 2;
 150   2              MEMORY$USERSTACK = BP;
 151   2              CALL RESTORE$EXECUTE$CODE$PTR(USERSTACK.OFF,USERSTACK.SEG);
 152   2           END;

 153   1        INTERRUPT1$ENTRY:
              /* THIS PROCEDURE IS CALLED WHEN THE CPU IS INTERRUPTED BY EXECUTING
                 AN INSTRUCTION WITH THE TRAP BIT SET (SINGLE STEP). */
              PROCEDURE INTERRUPT 1;
 154   2              USERSTACK.OFF = STACKPTR;          /* SAVE USER STACK INFO */
 155   2              USERSTACK.SEG = STACKBASE;
 156   2              STACKPTR =  MONITOR$STACKPTR;
 157   2              STACKBASE = MONITOR$STACKBASE;
 158   2              CALL SAVE$REGISTERS;
 159   2              FL = FL AND (NOT STEP$TRAP);        /* CLEAR STEP FLAG */
 160   2              IF LAST$COMMAND<>SS$COMMAND THEN
```

```
 161   2                CALL RESTORE$EXECUTE;              /* CONTINUE IF NOT SS */
 162   2             CALL KB$UPDATE$IP;
 163   2             IF CHAR=KBCOM THEN
 164   2                DO;
 165   3                IP = CSIP.OFF;
 166   3                CS = CSIP.SEG;
 167   3                FL = FL OR STEP$TRAP;              /* SET STEP FLAG */
 168   3                CALL RESTORE$EXECUTE;
 169   3                END;
 170   2             IF CHAR<>KBPER THEN  GOTO ERROR;
 172   2             GOTO AFTER$COMMAND;
 173   2          END;

 174   1       INTERRUPT3$ENTRY:
             /* THIS PROCEDURE IS CALLED WHEN THE CPU EXECUTES A 'INT 3' INSTRUCTION.
                THE MONITOR INSERTS THIS (0CCH) FOR A BREAKPOINT. ALSO AN EXTERNAL
                INTERRUPT OR A USER SOFTWARE INTERRUPT MAY CAUSE THIS PROCEDURE TO BE
                CALLED. */
             PROCEDURE INTERRUPT 3;
 175   2             USERSTACK.OFF = STACKPTR;          /* SAVE USER STACK INFO */
 176   2             USERSTACK.SEG = STACKBASE;
 177   2             STACKPTR =  MONITOR$STACKPTR;
 178   2             STACKBASE = MONITOR$STACKBASE;
 179   2             CALL SAVE$REGISTERS;
 180   2             GOTO AFTER$INTERRUPT;
 181   2          END;

 182   1       INIT$INT$VECTOR:
             /* THIS ROUTINE INITIALIZES AN INTERRRUPT VECTOR AS FOLLOWS: THE OFFSET
                FROM THE ADDRESS OF 'INT$ROUTINE' CORRECTED BY THE APPROPRIATE
                NUMBER OF BYTES FOR THE INTERRUPT PLM PROLOGUE. THE SEGMENT FROM THE
                CURRENT CS REGISTER IS DETERMINED BY A MACHINE LANGUAGE CODED
                SUBROUTINE. */
             PROCEDURE(INT$VECTOR$PTR,INT$ROUTINE$OFFSET);
 183   2             DECLARE INT$VECTOR$PTR POINTER, INT$ROUTINE$OFFSET WORD,
                      VECTOR BASED INT$VECTOR$PTR STRUCTURE (OFF WORD, SEG WORD),
                      CORRECTION LITERALLY '19H',  /* OFFSET FOR PROLOGUE */
                      INIT$INT$VECTOR$CODE(*) BYTE DATA
                      (055H,                             /* PUSH BP             */
                       08BH,0ECH,                        /* MOV BP,SP           */
                       08CH,0C8H,                        /* MOV AX,CS           */
                       0C4H,05EH,004H,                   /* LES BX,/BP/.PARM1   */
                       026H,089H,007H,                   /* MOV ES:W/BX/,AX     */
                       05DH,                             /* POP BP              */
                       0C2H,004H,000H),                  /* RET 4               */
                      INIT$INT$VECTOR$CODE$PTR WORD DATA (.INIT$INT$VECTOR$CODE);

 184   2             CALL INIT$INT$VECTOR$CODE$PTR(@VECTOR.SEG);    /* SEGMENT PORTION */
 185   2             VECTOR.OFF = INT$ROUTINE$OFFSET - CORRECTION;   /* OFFSET PORTION */
 186   2          END;


          /* ************************************************************************
             ************************************************************************

                                   COMMAND MODULE
                                   ==============
```

```
                ABSTRACT
                ========
                THIS MODULE CONTAINS ALL THE COMMANDS IMPLEMENTED AS INDIVIDUAL PROCEDURES
                AND CALLED FROM THE OUTER BLOCK OF THE COMMAND DISPATCH LOOP.

                MODULE ORGANIZATION
                ===================
                THIS MODULE CONTAINS THE FOLLOWING SECTIONS:

                   1. COMMANDS SECTION
                       KB$GO               GO
                       KB$SINGLE$STEP      SINGLE STEP
                       KB$EXAM$MEM         SUBSTITUTE MEMORY
                       KB$EXAM$REG         EXAMINE REGISTER
                       KB$MOVE             MOVE
                       KB$INPUT            INPUT PORT
                       KB$OUTPUT           OUTPUT PORT
                     2. COMMAND DISPATCH (OUTER BLOCK, MAIN PROGRAM LOOP)
                       NEXT$COMMAND         DISPATCH
                       ERROR                ERROR ROUTINE
             */

                /************************
                *   COMMANDS SECTION   *
                ************************/

187   1       KB$GO:
                /* IMPLEMENTS THE 'GO' COMMAND. DISPLAYS IP AND CURRENT INSTRUCTION
                   BYTE AND OPENS CS:IP FOR INPUT AND ONE OPTIONAL BREAKPOINT.
                   BEGINS EXECUTION WHEN A PERIOD IS DEPRESSED AND DISPLAYS 'E' IN
                   THE ADDRESS FIELD. UPON ENCOUNTERING A BREAKPOINT, 'BR' IS
                   DISPLAYED IN THE DATA FIELD. */
                PROCEDURE;
188   2            CALL KB$UPDATE$IP;                        /* OPTIONAL CHANGE CS:IP */
189   2            IF CHAR=KBCOM THEN
190   2               DO;                                    /* BREAKPOINT */
191   3               CALL KB$BLANK$ADDR$FIELD(1);
192   3               CALL KB$BLANK$DATA$FIELD(0);
193   3               CALL KB$GET$CHAR;
194   3               CALL KB$GET$ADDR(@BRK1,CSIP.SEG,1);
195   3               IF CHAR<>KBPER THEN  GOTO ERROR;
197   3               BRK1$SAVE = MEMORY$BRK1;
198   3               MEMORY$BRK1 = BREAK$INST;
199   3               IF MEMORY$BRK1<>BREAK$INST THEN  GOTO ERROR;
201   3               BRK1$FLAG = TRUE;
202   3               END;
                   ELSE
203   2               IF CHAR<>KBPER THEN  GOTO ERROR;
                   CALL KB$DISPLAY(@KB$EXEC,ADDR$FIELD,0);
206   2            CALL KB$BLANK$DATA$FIELD(0);
207   2            IP = CSIP.OFF;
208   2            CS = CSIP.SEG;
209   2            FL = FL AND (NOT STEP$TRAP);
210   2            CALL RESTORE$EXECUTE;
211   2            END;

212   1       KB$SINGLE$STEP:
```

```
                    /* IMPLEMENTS THE SINGLE STEP COMMAND. DISPLAYS IP AND THE
                       CURRENT INSTRUCTION WORD. OPENS CS:IP FOR INPUT. DEPRESSING
                       COMMA CAUSES THE MONITOR TO SINGLE STEP THE INSTRUCTION,
                       AND PERIOD TERMINATES THE COMMAND. */
                    PROCEDURE;
 213   2                CALL KB$UPDATE$IP;            /* OPTIONAL CHANGE OF CS:IP */
 214   2                IF CHAR<>KBCOM THEN  GOTO ERROR;
 216   2                IP = CSIP.OFF;
 217   2                CS = CSIP.SEG;
 218   2                FL = FL OR STEP$TRAP;         /* SET TRAP FLAG BIT IN PSW */
 219   2                CALL RESTORE$EXECUTE;
 220   2             END;

 221   1          KB$EXAM$MEM:
                    /* IMPLEMENTS THE EXAMINE MEMORY COMMAND. PROMPTS FOR AN
                       ADDRESS AND THEN DISPLAYS THE BYTE OR WORD AT THAT LOCATION.
                       IT THEN IS OPTIONALLY OPENED FOR INPUT. COMMA INCREMENTS TO
                       THE NEXT LOCATION. PERIOD TERMINATES. */
                    PROCEDURE;
 222   2                DECLARE W WORD;
 223   2                CALL KB$BLANK$ADDR$FIELD(1);          /* PROMPT FOR ADDRESS */
 224   2                CALL KB$GET$CHAR;
 225   2                CALL KB$GET$ADDR(@ARG1,CS,1);         /* GET ADDRESS */
 226   2                IF CHAR<>KBCOM THEN  GOTO ERROR;
 228   2                DO WHILE TRUE;
 229   3                   CALL KB$OUT$WORD(ARG1.OFF,ADDR$FIELD,0,BLANK); /* CLEAR PROMPT */
 230   3                   IF WORD$MODE THEN
 231   3                      CALL KB$OUT$WORD(MEMORY$WORD$ARG1,DATA$FIELD,1,NOBLANK);
                           ELSE
 232   3                      CALL KB$OUT$BYTE(MEMORY$ARG1,1);
 233   3                   CALL KB$GET$CHAR;
 234   3                   IF CHAR=KBPER THEN RETURN;
 236   3                   IF CHAR<>KBCOM THEN
 237   3                      IF WORD$MODE THEN
 238   3                         DO;
 239   4                         W = KB$GET$EXPR(DATA$FIELD,1,NOBLANK);
 240   4                         IF (CHAR<>KBCOM) AND (CHAR<>KBPER) THEN GOTO ERROR;
 242   4                         MEMORY$WORD$ARG1 = W;
 243   4                         IF MEMORY$WORD$ARG1<>W THEN  GOTO ERROR;
 245   4                         END;
                           ELSE
 246   3                         DO;
 247   4                         W = KB$GET$EXPR(DATA$BYTE,1,NOBLANK);
 248   4                         IF (CHAR<>KBCOM) AND (CHAR<>KBPER) THEN GOTO ERROR;
 250   4                         MEMORY$ARG1 = LOW(W);
 251   4                         IF MEMORY$ARG1<>LOW(W) THEN  GOTO ERROR;
 253   4                         END;
 254   3                   IF CHAR=KBPER THEN  RETURN;
 256   3                   IF WORD$MODE THEN
 257   3                      ARG1.OFF = ARG1.OFF + 2;
                           ELSE
 258   3                      ARG1.OFF = ARG1.OFF + 1;
 259   3                END;
 260   2             END;

 261   1          KB$EXAM$REG:
                    /* IMPLEMENTS THE EXAMINE REGISTER COMMAND. PROMPTS FOR A VALID
```

```
                  REGISTER KEY AND DISPLAYS THE VALUE OF THAT REGISTER WHICH IS
                  OPTIONALLY OPENED FOR INPUT. COMMA INCREMENTS TO NEXT REGISTER
                  UNLESS IT IS 'FL' WHICH TERMINATESS AS DOES PERIOD. */
               PROCEDURE;
262   2           DECLARE I BYTE, SAVE WORD;
263   2           CALL KB$BLANK$ADDR$FIELD(1);
264   2           CALL KB$GET$CHAR;
265   2           IF CHAR>0DH THEN  GOTO ERROR;            /* INVALID REG KEY */
267   2           I = CHAR;
268   2           DO WHILE TRUE;
269   3              DISP(0),DISP(1) = 0;                  /* DISPLAY REG NAME */
270   3              DISP(2) = KB$REG(I*2);
271   3              DISP(3) = KB$REG(I*2+1);
272   3              CALL KB$DISPLAY(@DISP,ADDR$FIELD,0);
273   3              CALL KB$OUT$WORD(REG$SAV(I),DATA$FIELD,1,NOBLANK); /* VALUE */
274   3              CALL KB$GET$CHAR;
275   3              IF CHAR<>KBCOM AND CHAR<>KBPER THEN
276   3                 DO;
277   4                 SAVE = KB$GET$EXPR(DATA$FIELD,1,NOBLANK); /* UPDATE */
278   4                 IF CHAR<>KBCOM AND CHAR<>KBPER THEN GOTO ERROR;
280   4                 REG$SAV(I) = SAVE;
281   4                 END;
282   3              IF CHAR=KBPER OR I=13 THEN  RETURN;              /* DONE? */
284   3              I = I + 1;
285   3           END;
286   2        END;

287   1     KB$MOVE:
            /* IMPLEMENTS THE MOVE COMMAND. PROMPTS FOR 3 ARGUMENTS AND MOVES
               THE BLOCK OF MEMORY SPECIFIED BY ARG1-ARG2 TO ARG3. IF THERE IS
               A DIFFERENCE WHEN READ BACK, THEN ERROR. */
            PROCEDURE;
288   2           CALL KB$BLANK$ADDR$FIELD(3);              /* FIRST ARGUMENT */
289   2           CALL KB$GET$CHAR;
290   2           CALL KB$GET$ADDR(@ARG1,CS,3);
291   2           IF CHAR<>KBCOM THEN  GOTO ERROR;
293   2           CALL KB$BLANK$ADDR$FIELD(2);              /* SECOND ARGUMENT */
294   2           CALL KB$GET$CHAR;
295   2           END$OFF = KB$GET$EXPR(ADDR$FIELD,2,BLANK);
296   2           IF END$OFF<ARG1.OFF THEN  GOTO ERROR;
298   2           IF CHAR<>KBCOM THEN  GOTO ERROR;
300   2           CALL KB$BLANK$ADDR$FIELD(1);              /* THIRD ARGUMENT */
301   2           CALL KB$GET$CHAR;
302   2           CALL KB$GET$ADDR(@ARG3,ARG1.SEG,1);
303   2           IF CHAR<>KBPER THEN  GOTO ERROR;
305   2     LOOP:
                  MEMORY$ARG3 = MEMORY$ARG1;
306   2           IF MEMORY$ARG3<>MEMORY$ARG1 THEN  GOTO ERROR;
308   2           IF ARG1.OFF = END$OFF THEN RETURN;
310   2           ARG1.OFF = ARG1.OFF + 1;
311   2           ARG3.OFF = ARG3.OFF + 1;
312   2           GOTO LOOP;
313   2        END;

314   1     KB$INPUT:
             /* PROMPTS FOR A PORT WORD IN THE ADDRESS FIELD. WHEN A COMMA
                IS ENTERED THE BYTE OR WORD IS DISPLAYED IN THE DATA FIELD.
```

```
                        THIS MAY BE REPEATED FOR MULTIPLE READING OF THE PORT. PERIOD
                        TERMINATES THE COMMAND. */
                     PROCEDURE;
 315    2              DECLARE PORT WORD;
 316    2              CALL KB$BLANK$ADDR$FIELD(1);                  /* PROMPT FOR PORT */
 317    2              CALL KB$GET$CHAR;
 318    2              PORT = KB$GET$EXPR(ADDR$FIELD,1,BLANK);       /* GET PORT NUMBER */
 319    2              CALL KB$OUT$WORD(PORT,ADDR$FIELD,0,BLANK);    /* REMOVE PROMPT */
 320    2        LOOP:
                       IF CHAR<>KBCOM THEN  GOTO ERROR;
 322    2              IF WORD$MODE THEN
 323    2                 CALL KB$OUT$WORD(INWORD(PORT),DATA$FIELD,0,NOBLANK);
                        ELSE
 324    2                 CALL KB$OUT$BYTE(INPUT(PORT),0);
 325    2              CALL KB$GET$CHAR;
 326    2              IF CHAR=KBPER THEN RETURN;
 328    2              GOTO LOOP;
 329    2           END;

 330    1        KB$OUTPUT:
                    /* PROMPTS FOR A PORT WORD IN THE ADDRESS FIELD AND A BYTE OR
                       WORD DATUM IN THE DATA FIELD. THIS DATUM IS OUTPUT A SINGLE TIME
                       TO THE SPECIFIED PORT. */
                    PROCEDURE;
 331    2              DECLARE (DATUM,PORT) WORD;
 332    2              CALL KB$BLANK$ADDR$FIELD(1);                  /* PROMPT FOR PORT */
 333    2              CALL KB$GET$CHAR;
 334    2              PORT = KB$GET$EXPR(ADDR$FIELD,1,BLANK);
 335    2              IF CHAR<>KBCOM THEN  GOTO ERROR;
 337    2              CALL KB$OUT$WORD(PORT,ADDR$FIELD,0,BLANK);    /* REMOVE PROMPT */
 338    2              CALL KB$BLANK$DATA$FIELD(1);                  /* PROMPT FOR DATUM */
 339    2              CALL KB$GET$CHAR;
 340    2        LOOP:
                       IF WORD$MODE THEN
 341    2                 DATUM = KB$GET$EXPR(DATA$FIELD,1,NOBLANK); /* GET DATUM WORD */
                        ELSE
 342    2                 DATUM = KB$GET$EXPR(DATA$BYTE,1,NOBLANK);  /* GET DATUM BYTE */
 343    2              IF CHAR=KBCOL THEN  GOTO ERROR;
 345    2              IF WORD$MODE THEN
 346    2                 OUTWORD(PORT) = DATUM;
                        ELSE
 347    2                 OUTPUT(PORT) = LOW(DATUM);
 348    2              IF CHAR=KBCOM THEN                  /* MULTIPLE OUTPUTS */
 349    2                 DO;
 350    3                 CALL KB$BLANK$DATA$FIELD(1);
 351    3                 CALL KB$GET$CHAR;
 352    3                 IF CHAR<>KBPER THEN GOTO LOOP;
 354    3                 END;
 355    2           END;


                    /*****************************************
                    *   COMMAND DISPATCH MAIN PROGRAM LOOP    *
                    *****************************************/

 356    1              DISABLE;
 357    1              OUTPUT(KB$STAT$PORT) = KB$INIT$MODE;      /* INIT 8279 */
```

```
 358   1              OUTPUT(KB$STAT$PORT) = KB$INIT$SCAN;
 359   1              CALL KB$DISPLAY(@KB$SIGNON,ADDR$FIELD,0); /* SIGN ON MESSAGE */
 360   1              CALL KB$DISPLAY(@KB$VERSION,DATA$FIELD,0); /* VERSION NUMBER */

                      /* INITIALIZE USER'S REGISTERS */
 361   1              CS,SS,DS,ES,FL,IP = 0;
 362   1              SP = USER$INIT$SP;

 363   1              CALL INIT$INT$VECTOR(@INT$VECTOR(1),.INTERRUPT1$ENTRY);
 364   1              CALL INIT$INT$VECTOR(@INT$VECTOR(2),.INTERRUPT3$ENTRY);
 365   1              CALL INIT$INT$VECTOR(@INT$VECTOR(3),.INTERRUPT3$ENTRY);
 366   1              BRK1$FLAG = FALSE;
 367   1              MONITOR$STACKBASE = STACKBASE; /* SAVE MONITOR STACK VALUES */
 368   1              MONITOR$STACKPTR = STACKPTR;
 369   1              GOTO AFTER$ERROR;

 370   1        NEXT$COMMAND:
                    /* THIS IS THE PERPETUAL COMMAND LOOP WHICH DISPATCHES TO EACH
                       COMMAND WHICH IS A SEPARATE PROCEDURE. */

                      CALL KB$DISPLAY(@KB$CMND$PMT,ADDR$FIELD,0);
 371   1        AFTER$ERROR:
                      CALL KB$GET$CHAR;
 372   1              CALL KB$BLANK$ADDR$FIELD(0);
 373   1              CALL KB$BLANK$DATA$FIELD(0);
 374   1              IF (LAST$COMMAND:=CHAR)>09H THEN  GOTO ERROR;
 376   1              WORD$MODE = FALSE;
 377   1              DO CASE CHAR;
 378   2                 CALL KB$EXAM$MEM;
 379   2                 CALL KB$EXAM$REG;
 380   2                 CALL KB$GO;
 381   2                 CALL KB$SINGLE$STEP;
 382   2                 CALL KB$INPUT;
 383   2                 CALL KB$OUTPUT;
 384   2                 CALL KB$MOVE;
 385   2                 DO;  WORD$MODE = TRUE; CALL KB$EXAM$MEM; END;
 389   2                 DO;  WORD$MODE = TRUE; CALL KB$INPUT;    END;
 393   2                 DO;  WORD$MODE = TRUE; CALL KB$OUTPUT;   END;
 397   2              END;

 398   1        AFTER$COMMAND:
                      CALL KB$BLANK$DATA$FIELD(0);
 399   1              GOTO NEXT$COMMAND;

 400   1        ERROR:
                    /* THIS ROUTINE HANDLES ALL ERRORS DETECTED BY THE MONITOR AND WILL
                       OUTPUT THE ERROR MESSAGE TO THE DISPLAY AND THEN JUMP TO
                       'AFTER$ERROR' TO GET ANOTHER COMMAND. */

                      CALL KB$DISPLAY(@KB$ERR$MSG,ADDR$FIELD,0);
 401   1              CALL KB$BLANK$DATA$FIELD(0);
 402   1              GOTO AFTER$ERROR;

 403   1        AFTER$INTERRUPT:
                    /* THIS ROUTINE CHECKS FOR THE LAST COMMAND WHEN THE MONITOR IS
                       REENTERED VIA THE INTERRUPT VECTOR. */
```

```
                    IF BRK1$FLAG THEN
 404   1                DO;
 405   2                MEMORY$BRK1 = BRK1$SAVE;
 406   2                BRK1$FLAG = FALSE;
 407   2                IF ((IP-1) AND 000FH)=(BRK1.OFF AND 000FH) AND
                             (SHR(IP-1,4)+CS)=(SHR(BRK1.OFF,4)+BRK1.SEG) THEN
 408   2                   DO;
 409   3                   IP = IP - 1;
 410   3                   CALL KB$DISPLAY(@KB$BRK1,DATA$FIELD,0);
 411   3                   GOTO NEXT$COMMAND;
 412   3                   END;
 413   2                END;
 414   1              GOTO AFTER$COMMAND;

 415   1           END MONITOR;     /* END OF MODULE */
               EOF
```

MODULE INFORMATION:

```
     CODE AREA SIZE      = 0B82H   2946D
     CONSTANT AREA SIZE  = 0000H      0D
     VARIABLE AREA SIZE  = 0065H    101D
     MAXIMUM STACK SIZE  = 0050H     80D
     865 LINES READ
     0 PROGRAM ERROR(S)

END OF PL/M-86 COMPILATION
```

SDK-86 KEYPAD MONITOR EXPANDED SOURCE LISTING


```
ISIS-II PL/M-86 V1.0 COMPILATION OF MODULE MONITOR
NO OBJECT MODULE REQUESTED
COMPILER INVOKED BY:  :F1:PLM86 :F1:KEYPAD.PLM LARGE OPTIMIZE(2) PAGEWIDTH(95) &
                      CODE XREF NOOBJECT PRINT(:F1:KEYPAD.PRT)


$TITLE('SDK86 KEYPAD MONITOR')
$NOINTVECTOR

 /* ************************************************************************
    ************************************************************************

        SDK-86 KEYPAD MONITOR, V1.1
        30 JUNE 1978

      (C) 1978 INTEL CORPORATION. ALL RIGHTS RESERVED. NO PART OF
    OF THIS PROGRAM OR PUBLICATION MAY BE REPRODUCED, TRANSMITTED,
    TRANSCRIBED, STORED IN A RETRIEVAL SYSTEM, OR TRANSLATED INTO ANY
    LANGUAGE, IN ANY FORM OR BY ANY MEANS, ELECTRONIC, MECHANICAL,
    MAGNETIC, OPTICAL, CHEMICAL, MANUAL OR OTHERWISE, WITHOUT THE PRIOR
    WRITTEN PERMISSION OF INTEL CORPORATION, 3065 BOWERS AVENUE, SANTA
    CLARA, CALIFORNIA, 95051.

    ************************************************************************
    ************************************************************************

    ABSTRACT
    ========
    THIS PROGRAM IS THE ROM BASED KEYPAD MONITOR FOR THE SDK-86.
    IT PROVIDES THE USER WITH A MODERATE LEVEL OF CAPABLILITY TO
    EXAMINIE/MODIFY MEMORY AND REGISTERS AND CONTROL PROGRAM EXECUTION.

    ENVIRONMENT
    ===========
    THE SDK KEYPAD MONITOR COMMUNICATES WITH THE USER VIA THE ONBOARD
    KEYBOARD AND 7 SEGMENT DISPLAY.

    PROGRAM ORGANIZATION
    ====================
    THE PROGRAM IS DIVIDED INTO 1 DATA AND 2 CODE MODULES:
       1. DATA DECLARATION MODULE.    GLOBAL DATA DECLARATIONS.
       2. COMMON ROUTINES.            LOWER LEVEL PROCEDURES
       3. COMMAND MODULE.             INDIVIDUAL COMMANDS AND OUTER BLOCK

    CALLING GRAPH
    =============
       >>COMMAND DISPATCH MODULE (OUTER BLOCK)
            INDIVIDUAL COMMAND PROCEDURES
               COMMON ROUTINES

    GLOBAL DATA STRUCTURES
    ======================
    THE MONITOR MAINTAINS THE USER'S MACHINE STATE (REGISTERS) IN A
    WORD ARRAY. THE REGISTERS ARE SAVED FROM THE USER'S STACK
    AS PUSHED BY PLM86 INTERRUPT PROCEDURE.
    POINTERS TO THE SDK-86 2**20 ADDRESS SPACE ARE IMPLEMENTED WITH
```

```
                POINTER STRUCTURES ALLOCATED AS 2 WORD STRUCTURES.

              */

  1             MONITOR:DO;                    /* BEGINNING OF MODULE */

              /* ***************************************************************
                 ***************************************************************

                                GLOBAL DATA DECLARATIONS MODULE
                                ===============================

                 ABSTRACT
                 ========
                 THIS MODULE CONTAINS ALL THE GLOBAL DATA DECLARATIONS AND
                 LITERALS (EQUATES).

                 MODULE ORGAINIZATION
                 ====================
                 THE MODULE IS DIVIDED INTO 5 SECTIONS:
                    1. UTILITY SECTION              GLOBAL FLAGS,VARIBLES,EQUATES
                    2. I/O SECTION                  I/O PORTS,MASKS,AND SPECIAL CHARS
                    3. MEMORY ARGUMENTS SECTIONS    STRUCTURES FOR MEDIUM POINTERS
                    4. REGISTER SECTION             USER REGISTER SAVE AREA
                    5. BOOT AND 8089 SECTION        BOOTSTRAP AND 8089 DESCRIPTOR
              */

                /************************
                 *    UTILITY SECTION   *
                 ************************/

  2   1       DECLARE
                    INT$VECTOR(5)     POINTER;              /* INTERRUPT VECTORS */
  3   1       DECLARE
                    MONITOR$STACKPTR  WORD,
                    MONITOR$STACKBASE WORD;                 /* MONITOR SS SAVE */

  4   1       DECLARE
                    COPYRIGHT(*) BYTE DATA ('(C) 1978 INTEL CORP');
  5   1       DECLARE
                    BRK1$FLAG       BYTE,                   /* TRUE IF BREAK SET */
                    BRK1$SAVE       BYTE,                   /* INST BREAK SAVE */
                    CHAR            BYTE,                   /* ONE CHAR LOOK AHEAD */
                    DISP(4)         BYTE,                   /* DISPLAY ARRAY  */
                    I               BYTE,
                    END$OFF         WORD,                     /* END OFFSET ADDRESS */
                    WORD$MODE       BYTE,
                    LAST$COMMAND    BYTE;
  6   1       DECLARE
                    TRUE            LITERALLY 'OFFH',
                    FALSE           LITERALLY '000H',
                    ADDR$FIELD      LITERALLY '1',          /* ADDR FIELD WORD OUTPUT */
                    DATA$FIELD      LITERALLY '0',          /* DATA FIELD WORD OUTPUT */
                    DATA$BYTE       LITERALLY '-1',         /* DATA FIELD BYTE OUTPUT */
                    BLANK           LITERALLY '1',
                    NOBLANK         LITERALLY '0',
                    BREAK$INST      LITERALLY '0CCH',       /* BREAKPOINT TRAP */
                    STEP$TRAP       LITERALLY '0100H',      /* SS TRAP FLAG MASK */
```

```
                  USER$INIT$SP    LITERALLY '100H',       /* USER STACK INITIAL */
                  GO$COMMAND      LITERALLY '2',          /* GO COMMAND CODE */
                  SS$COMMAND      LITERALLY '3';

               /*********************************
               *   I/O DECLARATIONS SECTION   *
               *********************************/

  7   1       DECLARE
                  KB$STAT$PORT    LITERALLY 'OFFEAH',          /* STATUS/COMMAND PORT */
                  KB$DATA$PORT    LITERALLY 'OFFE8H',          /* DATA PORT */
                  KB$INIT$MODE    LITERALLY '00H',             /* 8 8-BIT, LEFT ENTRY,
                                                                  ENCODE, 2 KEY LOCKOUT */
                  KB$INIT$SCAN    LITERALLY '39H',             /* 10MS SCAN RATE   */
                  KB$INRDY        LITERALLY '07H',             /* IN CHAR RDY MASK */
                  KB$CMND$PMT(*)  BYTE DATA (40H,00H,00H,00H), /* '-   ' */
                  KB$SIGNON(*)    BYTE DATA (40H,00H,7FH,7DH), /* '- 86' */
                  KB$VERSION(*)   BYTE DATA (00H,00H,86H,06H), /* '  1.1' */
                  KB$REG$PMT(*)   BYTE DATA (50H,00H,00H,00H), /* 'R   ' */
                  KB$ERR$MSG(*)   BYTE DATA (40H,79H,50H,50H), /* '-ERR' */
                  KB$EXEC(*)      BYTE DATA (79H,00H,00H,00H), /* 'E   ' */
                  KB$BRK1(*)      BYTE DATA (7CH,50H,00H,00H), /* 'BR  ' */
                  KB$BLANKS(*)    BYTE DATA (00H,00H,00H,00H), /* '    ' */
                  LED(*)          BYTE DATA
                     (3FH,06H,5BH,4FH,66H,6DH,7DH,07H,         /* '0'-'7' */
                      7FH,6FH,77H,7CH,39H,5EH,79H,71H);        /* '8'-'F' */
  8   1       DECLARE
                  KBPER           LITERALLY  '10H',            /* PERIOD          */
                  KBCOM           LITERALLY  '11H',            /* COMMA           */
                  KBREGKEY        LITERALLY  '15H',            /* REGISTER KEY    */
                  KBCOL           LITERALLY  '14H',            /* COLON (SEGMENT) */
                  KBPLUS          LITERALLY  '13H',            /* PLUS KEY        */
                  KBMINUS         LITERALLY  '12H';            /* MINUS KEY       */

               /***********************
               *   POINTER SECTION   *
               ***********************/

  9   1       DECLARE
                  MEMORY$ARG1$PTR  POINTER,                    /* ARGUMENT 1 */
                  ARG1 STRUCTURE (OFF WORD, SEG WORD)
                     AT (@MEMORY$ARG1$PTR),
                  MEMORY$ARG1 BASED MEMORY$ARG1$PTR BYTE,
                  MEMORY$WORD$ARG1 BASED MEMORY$ARG1$PTR WORD,

                  MEMORY$ARG3$PTR POINTER,                     /* ARGUMENT 3 */
                  ARG3 STRUCTURE (OFF WORD, SEG WORD)
                     AT (@MEMORY$ARG3$PTR),
                  MEMORY$ARG3 BASED MEMORY$ARG3$PTR BYTE,

                  MEMORY$BRK1$PTR POINTER,                     /* BREAKPOINT    */
                  BRK1 STRUCTURE (OFF WORD, SEG WORD)
                     AT (@MEMORY$BRK1$PTR),
                  MEMORY$BRK1 BASED MEMORY$BRK1$PTR BYTE,

                  MEMORY$CSIP$PTR POINTER,                     /* CS:IP INSTRUCTION */
                  CSIP STRUCTURE (OFF WORD, SEG WORD)
```

```
                    AT (@MEMORY$CSIP$PTR),
                 MEMORY$CSIP BASED MEMORY$CSIP$PTR BYTE,

                 MEMORY$USERSTACK$PTR POINTER,          /* USER'S STACK  */
                 USERSTACK STRUCTURE (OFF WORD, SEG WORD)
                    AT (@MEMORY$USERSTACK$PTR),
                 MEMORY$USERSTACK BASED MEMORY$USERSTACK$PTR WORD;


              /************************
              *   REGISTER SECTION   *
              ************************/

10  1       DECLARE
                 KB$REG(*)       BYTE DATA
                    (00H,77H,00H,7CH,00H,39H,00H,5EH,       /* AXBXCXDX   */
                     6DH,73H,7CH,73H,6DH,30H,5EH,30H,       /* SPBPSIDI   */
                     39H,6DH,5EH,6DH,6DH,6DH,79H,6DH,       /* CSDSSSES   */
                     30H,73H,71H,38H),                      /* IPFL       */
                 REG$SAV(14)     WORD,               /* USER'S SAVED REGS */
                 REG$ORD(*)      BYTE DATA
                    (7,6,1,3,2,0,9,11,12,8,13),     /* STACKED REG ORDER */
                 SP              LITERALLY 'REG$SAV( 4)',
                 BP              LITERALLY 'REG$SAV( 5)',
                 CS              LITERALLY 'REG$SAV( 8)',
                 DS              LITERALLY 'REG$SAV( 9)',
                 SS              LITERALLY 'REG$SAV(10)',
                 ES              LITERALLY 'REG$SAV(11)',
                 IP              LITERALLY 'REG$SAV(12)',
                 FL              LITERALLY 'REG$SAV(13)';


              /*************************************
              *   BOOTSTRAP JUMP AND 8089 VECTOR   *
              *************************************/

              /* THIS BOOT CONSISTS OF A LONG JUMP TO THE BEGINNING OF THE MONITOR
                 AT FF00+XXXX WHERE XXXX IS THE STARTING ADDRESS OFFSET (IP) AND
                 MUST BE DETERMINED AFTER EACH COMPILE. */
11  1       DECLARE
                 START$ADDR LITERALLY '009CH',              /* STARTING ADDRESS */
                 BOOT1(*)   BYTE AT (0FFFF0H) DATA (0EAH),  /* LONG JUMP OPCODE */
                 BOOT2(*)   WORD AT (0FFFF1H) DATA (START$ADDR),
                 BOOT3(*)   WORD AT (0FFFF3H) DATA (0FF00H); /* SEGMENT ADDRESS  */

              /* THIS TWO-WORD DATA IS A JUMP TO THE STARTING ADDRESS AND IS LOCATED
                 AT THE FIRST LOCATION OF ROM (NO OTHER DATA OR CONSTANT DECLARATIONS
                 MAY PRECEDE IT). THE JUMP IS ACTUALLY TO (START-ADDR)-4 SINCE THE
                 INSTRUCTION IS A RELATIVE JUMP OF LENGTH 3. */
12  1       DECLARE
                 BOOT4(*)   WORD DATA (0E990H,START$ADDR-4);  /* NOP,JMP START-ADDR */

              /* THIS BLOCK OF ROM AT FFFF6-FFFFA IS INITIALIZED FOR THE 8089
                  DEVICE AND POINTS TO A BLOCK OF RAM AT LOCATION 100H. */
13  1       DECLARE
                 BLOCK$8089      WORD    AT (0FFFF6H) DATA (00001H),
                 BLOCK$8089$PTR  POINTER AT (0FFFF8H) DATA (00100H);
```

```
            /* ************************************************************************
             ************************************************************************

                                COMMON PROCEDURES
                                =================
               ABSTRACT
               ========
               THIS MODULE CONTAINS THOSE LOWER LEVEL PROCEDURES CALLED BY HIGHER
               LEVEL ROUTINES.

               MODULE ORGANIZATION
               ===================
               THIS MODULE CONTAINS THE FOLLOWING SECTIONS:
                  1. BASIC I/O SECTION
                      KB$DISPLAY              DISPLAY TO LED FIELDS
                      KB$BLANK$DATA$FIELD     BLANK DATA FIELD WITH PROMPTS
                      KB$BLANK$ADDR$FIELD     BLANK ADDRESS FIELD WITH PROMPTS
                      KB$OUT$BYTE             OUTPUT A BYTE TO DISPLAY
                      KB$OUT$WORD             OUTPUT A WORD TO DISPLAY
                      KB$GET$CHAR             INPUT A CHAR FROM KEYPAD
                  2. ARGUMENT EXPRESSION EVALUATOR
                      KB$GET$EXPR             GET WORD EXPRESSION
                      KB$GET$ADDR             GET ADDRESS EXPRESSION
                      KB$UPDATE$IP            GET OPTIONAL CS:IP
                  3. INTERRRUPT AND RESTORE/EXECUTE ROUTINES
                      SAVE$REGISTERS          SAVES USERS REGISTERS
                      RESTORE$EXECUTE         RESTORE MACHINE STATE AND EXEC
                      INTERRUPR1$ENTRY        INTERRUPT FOR SINGLE STEP
                      INITERRUPT3$ENTRY       INTERRUPT ROUTINE FOR GO
                      INIT$INT$VECTOR         INITIALIZES INTERRRUPT VECTORS
            */

               /*************************
               *   BASIC I/O SECTION    *
               *************************/

14   1         KB$DISPLAY:
                                             ; STATEMENT # 14
               /* THIS ROUTINE DISPLAYS THE CONTENTS OF THE ARRAY POINTED TO BY
                  THE FIRST PARM TO THE FIELD SPECIFIED BY THE SECOND (ADDR OR
                  DATA). THE NUMBER OF DECIMAL POINTS OR PROMPTS IS DETERMINED
                  BY THE THIRD PARAMETER */
                   KBDISPLAY      PROC NEAR
          0251  55               PUSH    BP
          0252  8BEC             MOV     BP,SP
                PROCEDURE (PTR,FIELD,PROMPTS);
15   2             DECLARE PTR POINTER, (FIELD,PROMPTS,T) BYTE,
                      DISPLAY BASED PTR (1) BYTE;
16   2             IF FIELD=ADDR$FIELD THEN
                                             ; STATEMENT # 16
          0254  807E0601         CMP     [BP].FIELD,1H
          0258  7507             JNZ     @1
17   2                OUTPUT(KB$STAT$PORT) = 94H;              /* ADDRESS FIELD */
                                             ; STATEMENT # 17
          025A  BAEAFF           MOV     DX,OFFEAH
```

```
                  025D  B094            MOV     AL,94H
                  025F  EB05            JMP     @103
                                  @1:
                        ELSE
18   2                     OUTPUT(KB$STAT$PORT) = 90H;              /* DATA FIELD */
                                                        ; STATEMENT # 18
                  0261  BAEAFF          MOV     DX,0FFEAH
                  0264  B090            MOV     AL,90H
                                  @103:
                  0266  EE              OUT     DX
19   2                  DO I=0 TO 3;
                                                        ; STATEMENT # 19
                  0267  C6065F0000      MOV     I,0H
                                  @82:
                  026C  803E5F0003      CMP     I,3H
                  0271  7730            JA      @83
20   3                        T = DISPLAY(3-I);                      /* DISPLAY BACKWARDS! */
                                                        ; STATEMENT # 20
                  0273  B003            MOV     AL,3H
                  0275  2A065F00        SUB     AL,I
                  0279  B400            MOV     AH,0H
                  027B  89C6            MOV     SI,AX
                  027D  C45E08          LES     BX,[BP].PTR
                  0280  268A08          MOV     CL,ES:[BX].DISPLAY[SI]
                  0283  880E6200        MOV     T,CL
21   3                        IF PROMPTS>I THEN  T = T OR 80H;
                                                        ; STATEMENT # 21
                  0287  8A4604          MOV     AL,[BP].PROMPTS
                  028A  3A065F00        CMP     AL,I
                  028E  7605            JBE     @3
                                                        ; STATEMENT # 22
                  0290  800E620080      OR      T,80H
                                  @3:
23   3                        OUTPUT(KB$DATA$PORT) = T;
                                                        ; STATEMENT # 23
                  0295  BAE8FF          MOV     DX,0FFE8H
                  0298  A06200          MOV     AL,T
                  029B  EE              OUT     DX
24   3                  END;
                                                        ; STATEMENT # 24
                  029C  80065F0001      ADD     I,1H
                  02A1  75C9            JNZ     @82
                                  @83:
25   2               END;
                                                        ; STATEMENT # 25
                  02A3  5D              POP     BP
                  02A4  C20800          RET     8H
                        KBDISPLAY       ENDP

26   1            KB$BLANK$DATA$FIELD:
                                                        ; STATEMENT # 26
                     /* THIS ROUTINE BLANKS THE DATA FIELD OF THE DISPLAY WITH THE
                        NUMBER OF PROMPTS (DECIMAL POINTS) AS SPECIFIED BY THE PARAMETER. */
                        KBBLANKDATAFIELD      PROC NEAR
                  02A7  55              PUSH    BP
                  02A8  8BEC            MOV     BP,SP
                     PROCEDURE(PROMPTS);
```

```
 27   2              DECLARE PROMPTS BYTE;
 28   2              CALL KB$DISPLAY(@KB$BLANKS,DATA$FIELD,PROMPTS);
                                                  ; STATEMENT # 28
          02AA  2E8D063700      LEA     AX,CS:KBBLANKS
          02AF  0E              PUSH    CS        ; 1
          02B0  50              PUSH    AX        ; 2
          02B1  B000            MOV     AL,0H
          02B3  50              PUSH    AX        ; 3
          02B4  FF7604          PUSH    [BP].PROMPTS; 4
          02B7  E897FF          CALL    KBDISPLAY
 29   2           END;
                                                  ; STATEMENT # 29
          02BA  5D              POP     BP
          02BB  C20200          RET     2H
                    KBBLANKDATAFIELD        ENDP

 30   1        KB$BLANK$ADDR$FIELD:
                                                  ; STATEMENT # 30
                /* THIS PROCEDURE BLANKS THE ADDRESS FIELD OF THE DISPALY WITH THE
                   NUMBER OF PROMPTS SPECIFIED BY THE PARAMETER 'PROMPTS'. */
                    KBBLANKADDRFIELD        PROC NEAR
          02BE  55              PUSH    BP
          02BF  8BEC            MOV     BP,SP
           PROCEDURE(PROMPTS);
 31   2              DECLARE PROMPTS BYTE;
 32   2              CALL KB$DISPLAY(@KB$BLANKS,ADDR$FIELD,PROMPTS);
                                                  ; STATEMENT # 32
          02C1  2E8D063700      LEA     AX,CS:KBBLANKS
          02C6  0E              PUSH    CS        ; 1
          02C7  50              PUSH    AX        ; 2
          02C8  B001            MOV     AL,1H
          02CA  50              PUSH    AX        ; 3
          02CB  FF7604          PUSH    [BP].PROMPTS; 4
          02CE  E880FF          CALL    KBDISPLAY
 33   2           END;
                                                  ; STATEMENT # 33
          02D1  5D              POP     BP
          02D2  C20200          RET     2H
                    KBBLANKADDRFIELD        ENDP

 34   1        KB$OUT$BYTE:
                                                  ; STATEMENT # 34
                /* THIS ROUTINE OUTPUTS THE BYTE INPUT FROM THE FIRST PARAMETER
                   TO THE DATA FIELD WITH THE NUMBER OF PROMPTS SPECIFIED BY THE
                   SECOND PARAMETER. */
                    KBOUTBYTE       PROC NEAR
          02D5  55              PUSH    BP
          02D6  8BEC            MOV     BP,SP
           PROCEDURE(B,PROMPTS);
 35   2              DECLARE (B,PROMPTS) BYTE;
 36   2              DISP(0),DISP(1) = 0;                    /* FIRST TWO BLANK */
                                                  ; STATEMENT # 36
          02D8  B000            MOV     AL,0H
          02DA  A25B00          MOV     DISP,AL
          02DD  A25C00          MOV     DISP+1H,AL
 37   2              DISP(2) = LED(SHR(B,4) AND 0FH);
                                                  ; STATEMENT # 37
```

```
          02E0  8A5E06          MOV     BL,[BP].B
          02E3  B104            MOV     CL,4H
          02E5  D2EB            SHR     BL,CL
          02E7  80E30F          AND     BL,0FH
          02EA  B700            MOV     BH,0H
          02EC  2E8A4F3B        MOV     CL,CS:LED[BX]
          02F0  880E5D00        MOV     DISP+2H,CL
38   2              DISP(3) = LED(B AND 0FH);
                                                ; STATEMENT # 38
          02F4  8A5E06          MOV     BL,[BP].B
          02F7  80E30F          AND     BL,0FH
          02FA  B700            MOV     BH,0H
          02FC  2E8A4F3B        MOV     CL,CS:LED[BX]
          0300  880E5E00        MOV     DISP+3H,CL
39   2              CALL KB$DISPLAY(@DISP,DATA$FIELD,PROMPTS);
                                                ; STATEMENT # 39
          0304  8D0E5B00        LEA     CX,DISP
          0308  1E              PUSH    DS      ; 1
          0309  51              PUSH    CX      ; 2
          030A  50              PUSH    AX      ; 3
          030B  FF7604          PUSH    [BP].PROMPTS; 4
          030E  E840FF          CALL    KBDISPLAY
40   2             END;
                                                ; STATEMENT # 40
          0311  5D              POP     BP
          0312  C20400          RET     4H
                          KBOUTBYTE       ENDP

41   1        KB$OUT$WORD:
                                                ; STATEMENT # 41
                  /* THIS ROUTINE OUTPUTS THE FIRST PARM TO THE FIELD SPECIFIED
                     BY THE SECOND WITH THE NUMBER OF PROMPTS SPECIFIED BY THE THIRD.
                     LEADING ZERO BLANKING IS PERFORMED IF SPECIFIED BY THE FOURTH
                     PARAMETER. */
                          KBOUTWORD       PROC NEAR
          0315  55              PUSH    BP
          0316  8BEC            MOV     BP,SP
                   PROCEDURE(W,FIELD,PROMPTS,BLANKING);
42   2              DECLARE W WORD, (FIELD,PROMPTS,BLANKING) BYTE;
43   2              DO I=0 TO 3;
                                                ; STATEMENT # 43
          0318  C6065F0000      MOV     I,0H
                          @84:
          031D  803E5F0003      CMP     I,3H
          0322  7729            JA      @85
44   3                   DISP(I) = LED(SHR(W,(3-I)*4) AND 000FH);
                                                ; STATEMENT # 44
          0324  B003            MOV     AL,3H
          0326  2A065F00        SUB     AL,I
          032A  B104            MOV     CL,4H
          032C  F6E1            MUL     CL
          032E  89C1            MOV     CX,AX
          0330  8B5E0A          MOV     BX,[BP].W
          0333  D3EB            SHR     BX,CL
          0335  81E30F00        AND     BX,0FH
          0339  2E8A473B        MOV     AL,CS:LED[BX]
          033D  8A1E5F00        MOV     BL,I
```

```
                0341  B700            MOV     BH,0H
                0343  88475B          MOV     DISP[BX],AL
45   3              END;
                                              ; STATEMENT # 45
                0346  80065F0001      ADD     I,1H
                034B  75D0            JNZ     @84
                        @85:
46   2              IF BLANKING=BLANK THEN                  /* BLANK LEADING 0'S */
                                              ; STATEMENT # 46
                034D  807E0401        CMP     [BP].BLANKING,1H
                0351  7535            JNZ     @4
47   2                    DO;
48   3                    I = 0;
                                              ; STATEMENT # 48
                0353  C6065F0000      MOV     I,0H
49   3                    DO WHILE DISP(I)=3FH AND I<3;
                                              ; STATEMENT # 49
                        @86:
                0358  8A1E5F00        MOV     BL,I
                035C  B700            MOV     BH,0H
                035E  807F5B3F        CMP     DISP[BX],3FH
                0362  B0FF            MOV     AL,0FFH
                0364  7401            JZ      $+3H
                0366  40              INC     AX
                0367  50              PUSH    AX      ; 1
                0368  80FB03          CMP     BL,3H
                036B  B0FF            MOV     AL,0FFH
                036D  7201            JB      $+3H
                036F  40              INC     AX
                0370  59              POP     CX      ; 1
                0371  22C1            AND     AL,CL
                0373  D0D8            RCR     AL,1
                0375  7311            JNB     @4
50   4                  DISP(I) = 0;
                                              ; STATEMENT # 50
                0377  8A1E5F00        MOV     BL,I
                037B  B700            MOV     BH,0H
                037D  C6475B00        MOV     DISP[BX],0H
51   4                  I = I + 1;
                                              ; STATEMENT # 51
                0381  80065F0001      ADD     I,1H
52   4                END;
                                              ; STATEMENT # 52
                0386  EBD0            JMP     @86
53   3                END;
                        @4:
54   2              CALL KB$DISPLAY(@DISP,FIELD,PROMPTS);
                                              ; STATEMENT # 54
                0388  8D065B00        LEA     AX,DISP
                038C  1E              PUSH    DS      ; 1
                038D  50              PUSH    AX      ; 2
                038E  FF7608          PUSH    [BP].FIELD; 3
                0391  FF7606          PUSH    [BP].PROMPTS; 4
                0394  E8BAFE          CALL    KBDISPLAY
55   2            END;
                                              ; STATEMENT # 55
                0397  5D              POP     BP
```

```
                0398  C20800          RET     8H
                          KBOUTWORD     ENDP

 56   1       KB$GET$CHAR:
                                              ; STATEMENT # 56
                  /* READS ONE CHARACTER FROM THE FIFO OF THE 8279. WAITS UNTIL
                     CHARACTER IS AVAILABLE AND THEN RETURNS THE CHARACTER IN GLOBAL
                     VARIABLE 'CHAR'. */
                          KBGETCHAR     PROC NEAR
                0399  55              PUSH    BP
                039C  8BEC            MOV     BP,SP
                   PROCEDURE;
 57   2                DO WHILE (INPUT(KB$STAT$PORT) AND KB$INRDY)=0; END;
                                              ; STATEMENT # 57
                          @88:
                039E  BAEAFF          MOV     DX,OFFEAH
                03A1  EC              IN      DX
                03A2  A807            TEST    AL,7H
                03A4  74F8            JZ      @88
 59   2                OUTPUT(KB$STAT$PORT) = 040H;          /* ENABLE INPUT DATA */
                                              ; STATEMENT # 59
                03A6  BAEAFF          MOV     DX,OFFEAH
                03A9  B040            MOV     AL,40H
                03AB  EE              OUT     DX
 60   2                CHAR = INPUT(KB$DATA$PORT);           /* READ CHARACTER */
                                              ; STATEMENT # 60
                03AC  BAE8FF          MOV     DX,OFFE8H
                03AF  EC              IN      DX
                03B0  A25A00          MOV     CHAR,AL
 61   2             END;
                                              ; STATEMENT # 61
                03B3  5D              POP     BP
                03B4  C3              RET
                          KBGETCHAR     ENDP


                  /*********************************************
                  *   ARGUMENT EXPRESSION EVALUATOR SECTION   *
                  *********************************************/

 62   1       KB$GET$EXPR:
                                              ; STATEMENT # 62
                  /* THIS ROUTINE GATHERS CHARACTERS FROM THE INPUT STREAM
                     AND FORMS A WORD EXPRESSION WHICH IS RETURNED AS THE VALUE
                     OF THE PROCEDURE. THE CHARACTERS INPUT ARE ECHOED TO THE
                     ADDRESS OR DATA FIELD AS SPECIFIED BY THE FIRST PARAMETER.
                     IF THE FIRST PARM IS 0 (ADDR) OR 1 (DATA) THEN THE EXPRESSION IS
                     OUTPUT AS A WORD AND IF -1 (DATA$BYTE) THEN AS A BYTE ALWAYS IN
                     DATA FIELD. THE NUMBER OF PROMPTS BY THE SECOND PARM, AND LEADING
                     ZERO BLANKING BY THE THIRD.  EXPRESSION ARE TERMINATED WITH
                     A COMMA,PERIOD OR COLON. 'CHAR' WILL CONTAIN ONE OF THESE
                     ON EXIT. */
                          KBGETEXPR     PROC NEAR
                03B5  55              PUSH    BP
                03B6  8BEC            MOV     BP,SP
                   PROCEDURE (FIELD,PROMPTS,BLANKING) WORD;
 63   2                DECLARE (FIELD,PROMPTS,BLANKING) BYTE,
```

```
                    (SAVE,W) WORD, OPER BYTE;
64   2                OPER = KBPLUS;
                                                ; STATEMENT # 64
          03B8  C606630013      MOV     OPER,13H
65   2                W = 0;
                                                ; STATEMENT # 65
          03BD  C7064C000000    MOV     W,0H
66   2                DO WHILE TRUE;
                                                ; STATEMENT # 66
                        @90:
67   3                   IF CHAR=KBREGKEY THEN
                                                ; STATEMENT # 67
          03C3  803E5A0015      CMP     CHAR,15H
          03C8  7560            JNZ     @5
68   3                      DO;                              /* REGISTER NAME */
69   4                      CALL KB$DISPLAY(@KB$REG$PMT,FIELD,PROMPTS);
                                                ; STATEMENT # 69
          03CA  2E8D062700      LEA     AX,CS:KBREGPMT
          03CF  0E              PUSH    CS      ; 1
          03D0  50              PUSH    AX      ; 2
          03D1  FF7608          PUSH    [BP].FIELD; 3
          03D4  FF7606          PUSH    [BP].PROMPTS; 4
          03D7  E877FE          CALL    KBDISPLAY
70   4                      CALL KB$GET$CHAR;
                                                ; STATEMENT # 70
          03DA  E8BEFF          CALL    KBGETCHAR
71   4                      IF CHAR>0DH THEN  GOTO ERROR;     /* INVALID REG KEY */
                                                ; STATEMENT # 71
          03DD  803E5A000D      CMP     CHAR,0DH
          03E2  7603            JBE     $+5H
          03E4  E9D5FD          JMP     ERROR
73   4                      SAVE = REG$SAV(CHAR);
                                                ; STATEMENT # 73
          03E7  8A1E5A00        MOV     BL,CHAR
          03EB  B700            MOV     BH,0H
          03ED  D1E3            SHL     BX,1
          03EF  8B472E          MOV     AX,REGSAV[BX]
          03F2  A34A00          MOV     SAVE,AX
74   4                      IF FIELD=DATA$BYTE THEN
                                                ; STATEMENT # 74
          03F5  807E08FF        CMP     [BP].FIELD,0FFH
          03F9  750C            JNZ     @7
75   4                         CALL KB$OUT$BYTE(LOW(SAVE),PROMPTS);
                                                ; STATEMENT # 75
          03FB  A14A00          MOV     AX,SAVE
          03FE  50              PUSH    AX      ; 1
          03FF  FF7606          PUSH    [BP].PROMPTS; 2
          0402  E8D0FE          CALL    KBOUTBYTE
          0405  EB10            JMP     @8
                          @7:
                        ELSE
76   4                         CALL KB$OUT$WORD(SAVE,FIELD,PROMPTS,BLANKING);
                                                ; STATEMENT # 76
          0407  FF364A00        PUSH    SAVE    ; 1
          040B  FF7608          PUSH    [BP].FIELD; 2
          040E  FF7606          PUSH    [BP].PROMPTS; 3
          0411  FF7604          PUSH    [BP].BLANKING; 4
```

```
                0414  E8FEFE          CALL    KBOUTWORD
                                @8:
 77    4                    CALL KB$GET$CHAR;
                                                ; STATEMENT # 77
                0417  E881FF          CALL    KBGETCHAR
 78    4                    END;
                                                ; STATEMENT # 78
                                @6:
                041A  803E630013      CMP     OPER,13H
                041F  7559            JNZ     @13
                0421  A14A00          MOV     AX,SAVE
                0424  01064C00        ADD     W,AX
                0428  EB57            JMP     @14
                                @5:
                               ELSE
 79    3                    DO;                                 /* NUMBER */
 80    4                    IF CHAR>0FH THEN  GOTO ERROR;       /* INVALID DIGIT */
                                                ; STATEMENT # 80
                042A  803E5A000F      CMP     CHAR,0FH
                042F  7603            JBE     $+5H
                0431  E988FD          JMP     ERROR
 82    4                    SAVE = 0;
                                                ; STATEMENT # 82
                0434  C7064A000000    MOV     SAVE,0H
 83    4                    DO WHILE CHAR<=0FH;
                                                ; STATEMENT # 83
                                @92:
                043A  803E5A000F      CMP     CHAR,0FH
                043F  77D9            JA      @6
 84    5                      SAVE = SHL(SAVE,4) + DOUBLE(CHAR);
                                                ; STATEMENT # 84
                0441  A14A00            MOV     AX,SAVE
                0444  B104              MOV     CL,4H
                0446  D3E0              SHL     AX,CL
                0448  8A0E5A00          MOV     CL,CHAR
                044C  B500              MOV     CH,0H
                044E  03C1              ADD     AX,CX
                0450  A34A00            MOV     SAVE,AX
 85    5                      IF FIELD=DATA$BYTE THEN
                                                ; STATEMENT # 85
                0453  807E08FF          CMP     [BP].FIELD,0FFH
                0457  750C              JNZ     @11
 86    5                        CALL KB$OUT$BYTE(LOW(SAVE),PROMPTS);
                                                ; STATEMENT # 86
                0459  A14A00            MOV     AX,SAVE
                045C  50                PUSH    AX      ; 1
                045D  FF7606            PUSH    [BP].PROMPTS; 2
                0460  E872FE            CALL    KBOUTBYTE
                0463  EB10              JMP     @12
                                @11:
                                     ELSE
 87    5                        CALL KB$OUT$WORD(SAVE,FIELD,PROMPTS,BLANKING);
                                                ; STATEMENT # 87
                0465  FF364A00          PUSH    SAVE    ; 1
                0469  FF7608            PUSH    [BP].FIELD; 2
                046C  FF7606            PUSH    [BP].PROMPTS; 3
                046F  FF7604            PUSH    [BP].BLANKING; 4
```

```
                0472  E8A0FE          CALL    KBOUTWORD
                              @12:
 88    5                        CALL KB$GET$CHAR;
                                                ; STATEMENT # 88
                0475  E823FF          CALL    KBGETCHAR
 89    5                      END;
                                                ; STATEMENT # 89
                0478  EBC0            JMP     @92
 90    4                      END;
 91    3                    IF OPER=KBPLUS THEN                    /* EVAL PREV OPER */
 92    3                      W = W + SAVE;
                                                ; STATEMENT # 92
                             @13:
                            ELSE
 93    3                      W = W - SAVE;
                                                ; STATEMENT # 93
                047A  A14A00          MOV     AX,SAVE
                047D  29064C00        SUB     W,AX
                             @14:
 94    3                    IF FIELD=DATA$BYTE THEN
                                                ; STATEMENT # 94
                0481  807E08FF        CMP     [BP].FIELD,0FFH
                0485  750C            JNZ     @15
 95    3                      CALL KB$OUT$BYTE(LOW(W),PROMPTS);
                                                ; STATEMENT # 95
                0487  A14C00          MOV     AX,W
                048A  50              PUSH    AX      ; 1
                048B  FF7606          PUSH    [BP].PROMPTS; 2
                048E  E844FE          CALL    KBOUTBYTE
                0491  EB10            JMP     @16
                             @15:
                            ELSE
 96    3                      CALL KB$OUT$WORD(W,FIELD,PROMPTS,BLANKING);
                                                ; STATEMENT # 96
                0493  FF364C00        PUSH    W       ; 1
                0497  FF7608          PUSH    [BP].FIELD; 2
                049A  FF7606          PUSH    [BP].PROMPTS; 3
                049D  FF7604          PUSH    [BP].BLANKING; 4
                04A0  E872FE          CALL    KBOUTWORD
                             @16:
 97    3                    IF CHAR=KBCOM OR CHAR=KBPER OR CHAR=KBCOL THEN
                                                ; STATEMENT # 97
                04A3  A05A00          MOV     AL,CHAR
                04A6  3C11            CMP     AL,11H
                04A8  B0FF            MOV     AL,0FFH
                04AA  7401            JZ      $+3H
                04AC  40              INC     AX
                04AD  50              PUSH    AX      ; 1
                04AE  803E5A0010      CMP     CHAR,10H
                04B3  B0FF            MOV     AL,0FFH
                04B5  7401            JZ      $+3H
                04B7  40              INC     AX
                04B8  59              POP     CX      ; 1
                04B9  0AC1            OR      AL,CL
                04BB  50              PUSH    AX      ; 1
                04BC  803E5A0014      CMP     CHAR,14H
                04C1  B0FF            MOV     AL,0FFH
```

```
                04C3  7401            JZ      $+3H
                04C5  40              INC     AX
                04C6  59              POP     CX      ; 1
                04C7  0AC1            OR      AL,CL
                04C9  D0D8            RCR     AL,1
                04CB  7305            JNB     @17
 98   3                    RETURN W;
                                                ; STATEMENT # 98
                04CD  A14C00          MOV     AX,W
                04D0  EB2B            JMP     @2
                          @17:
 99   3                  IF CHAR=KBPLUS OR CHAR=KBMINUS THEN
                                                ; STATEMENT # 99
                04D2  803E5A0013      CMP     CHAR,13H
                04D7  B0FF            MOV     AL,0FFH
                04D9  7401            JZ      $+3H
                04DB  40              INC     AX
                04DC  50              PUSH    AX      ; 1
                04DD  803E5A0012      CMP     CHAR,12H
                04E2  B0FF            MOV     AL,0FFH
                04E4  7401            JZ      $+3H
                04E6  40              INC     AX
                04E7  59              POP     CX      ; 1
                04E8  0AC1            OR      AL,CL
                04EA  D0D8            RCR     AL,1
                04EC  7203            JB      $+5H
                04EE  E9CBFC          JMP     ERROR
100   3                    OPER = CHAR;
                                                ; STATEMENT # 100
                04F1  A05A00          MOV     AL,CHAR
                04F4  A26300          MOV     OPER,AL
                        ELSE
101   3                    GOTO ERROR;
102   3                  CALL KB$GET$CHAR;                    /* GET NEXT CHAR */
                                                ; STATEMENT # 102
                04F7  E8A1FE          CALL    KBGETCHAR
103   3              END;
                                                ; STATEMENT # 103
                04FA  E9C6FE          JMP     @90
                          @91:
104   2           END;
                                                ; STATEMENT # 104
                          @2:
                04FD  5D              POP     BP
                04FE  C20600          RET     6H
                      KBGETEXPR       ENDP

105   1       KB$GET$ADDR:
                                                ; STATEMENT # 105
                  /* THIS ROUTINE GATHERS CHARACTERS FROM THE INPUT STREAM AND
                     FORMS AN ADDRESS EXPRESSION OF SEGMENT PART AND OFFSET PART.
                     THE FIRST CHARACTER HAS ALREADY BEEN READ INTO GLOBAL 'CHAR'.
                     IF NO SEGMENT IS ENTERED, THE SECOMD PARM DEFAULT IS USED, THE
                     ADDRESS EXPRESSION IS DISPLAYED IN THE ADDRESS FIELD WITH THE
                     NUMBER OF PROMPTS SPECIFIED BY THE THIRD PARM. */
                      KBGETADDR       PROC NEAR
                0501  55              PUSH    BP
```

```
                0502  8BEC            MOV     BP,SP
                       PROCEDURE(PTR,DEFAULT$BASE,PROMPTS);
106   2                   DECLARE PTR POINTER, DEFAULT$BASE WORD, PROMPTS BYTE,
                             ARG BASED PTR STRUCTURE (OFF WORD, SEG WORD);
107   2                   ARG.SEG = DEFAULT$BASE;
                                                    ; STATEMENT # 107
                0504  8B4606          MOV     AX,[BP].DEFAULTBASE
                0507  C45E08          LES     BX,[BP].PTR
                050A  26894702        MOV     ES:ARG[BX+2H],AX
108   2                   ARG.OFF = KB$GET$EXPR(ADDR$FIELD,PROMPTS,BLANK);
                                                    ; STATEMENT # 108
                050E  B001            MOV     AL,1H
                0510  50              PUSH    AX      ; 1
                0511  FF7604          PUSH    [BP].PROMPTS; 2
                0514  50              PUSH    AX      ; 3
                0515  E89DFE          CALL    KBGETEXPR
                0518  C45E08          LES     BX,[BP].PTR
                051B  268907          MOV     ES:ARG[BX],AX
109   2                   IF CHAR=KBCOL THEN                    /* SEGMENT SPEC'D */
                                                    ; STATEMENT # 109
                051E  803E5A0014      CMP     CHAR,14H
                0523  7527            JNZ     @20
110   2                       DO;
111   3                       CALL KB$GET$CHAR;
                                                    ; STATEMENT # 111
                0525  E873FE          CALL    KBGETCHAR
112   3                       ARG.SEG = ARG.OFF;
                                                    ; STATEMENT # 112
                0528  C45E08          LES     BX,[BP].PTR
                052B  268B07          MOV     AX,ES:ARG[BX]
                052E  26894702        MOV     ES:ARG[BX+2H],AX
113   3                       ARG.OFF = KB$GET$EXPR(ADDR$FIELD,PROMPTS,BLANK);
                                                    ; STATEMENT # 113
                0532  B001            MOV     AL,1H
                0534  50              PUSH    AX      ; 1
                0535  FF7604          PUSH    [BP].PROMPTS; 2
                0538  50              PUSH    AX      ; 3
                0539  E879FE          CALL    KBGETEXPR
                053C  C45E08          LES     BX,[BP].PTR
                053F  268907          MOV     ES:ARG[BX],AX
114   3                       IF CHAR=KBCOL THEN  GOTO ERROR;
                                                    ; STATEMENT # 114
                0542  803E5A0014      CMP     CHAR,14H
                0547  7503            JNZ     $+5H
                0549  E970FC          JMP     ERROR
                                                    ; STATEMENT # 115
116   3                       END;
                                @20:
117   2                 END;
                                                    ; STATEMENT # 117
                054C  5D              POP     BP
                054D  C20800          RET     8H
                           KBGETADDR      ENDP

118   1             KB$UPDATE$IP:
                                                    ; STATEMENT # 118
                      /* THIS ROUTINE IS CALLED BY SINGLE STEP AND GO TO OUTPUT THE CURRENT
```

```
                    CS:IP AND THE CURRENT INSTRUCTION BYTE.  CS:IP IS OPENED FOR
                    OPTIONAL INPUT. */
                        KBUPDATEIP      PROC NEAR
          0550  55                  PUSH    BP
          0551  8BEC                MOV     BP,SP
                 PROCEDURE;
119   2             CALL KB$OUT$WORD(IP,ADDR$FIELD,1,BLANK);    /* DISPLAY IP */
                                                ; STATEMENT # 119
          0553  FF364600            PUSH    REGSAV+18H; 1
          0557  B001                MOV     AL,1H
          0559  50                  PUSH    AX      ; 2
          055A  50                  PUSH    AX      ; 3
          055B  50                  PUSH    AX      ; 4
          055C  E8B6FD              CALL    KBOUTWORD
120   2             CSIP.OFF = IP;
                                                ; STATEMENT # 120
          055F  A14600              MOV     AX,REGSAV+18H
          0562  A32600              MOV     CSIP,AX
121   2             CSIP.SEG = CS;
                                                ; STATEMENT # 121
          0565  A13E00              MOV     AX,REGSAV+10H
          0568  A32800              MOV     CSIP+2H,AX
122   2             CALL KB$OUT$BYTE(MEMORY$CSIP,0);
                                                ; STATEMENT # 122
          056B  C41E2600            LES     BX,MEMORYCSIPPTR
          056F  26FF37              PUSH    ES:MEMORYCSIP[BX]
          0572  B000                MOV     AL,0H
          0574  50                  PUSH    AX      ; 2
          0575  E85DFD              CALL    KBOUTBYTE
123   2             CALL KB$GET$CHAR;
                                                ; STATEMENT # 123
          0578  E820FE              CALL    KBGETCHAR
124   2             IF CHAR<>KBCOM AND CHAR<>KBPER THEN
                                                ; STATEMENT # 124
          057B  803E5A0011          CMP     CHAR,11H
          0580  B0FF                MOV     AL,0FFH
          0582  7501                JNZ     $+3H
          0584  40                  INC     AX
          0585  50                  PUSH    AX      ; 1
          0586  803E5A0010          CMP     CHAR,10H
          058B  B0FF                MOV     AL,0FFH
          058D  7501                JNZ     $+3H
          058F  40                  INC     AX
          0590  59                  POP     CX      ; 1
          0591  22C1                AND     AL,CL
          0593  D0D8                RCR     AL,1
          0595  731C                JNB     @22
125   2                 DO;                                   /* CHANGE CS:IP */
126   3                 CALL KB$BLANK$ADDR$FIELD(1);
                                                ; STATEMENT # 126
          0597  B001                MOV     AL,1H
          0599  50                  PUSH    AX      ; 1
          059A  E821FD              CALL    KBBLANKADDRFIELD
127   3                 CALL KB$BLANK$DATA$FIELD(0);
                                                ; STATEMENT # 127
          059D  B000                MOV     AL,0H
          059F  50                  PUSH    AX      ; 1
```

```
                05A0  E804FD          CALL    KBBLANKDATAFIELD
128     3             CALL KB$GET$ADDR(@CSIP,CS,1);
                                              ; STATEMENT # 128
                05A3  8D062600        LEA     AX,CSIP
                05A7  1E              PUSH    DS      ; 1
                05A8  50              PUSH    AX      ; 2
                05A9  FF363E00        PUSH    REGSAV+10H; 3
                05AD  B001            MOV     AL,1H
                05AF  50              PUSH    AX      ; 4
                05B0  E84EFF          CALL    KBGETADDR
129     3             END;
                       @22:
130     2           END;
                                              ; STATEMENT # 130
                05B3  5D              POP     BP
                05B4  C3              RET
                      KBUPDATEIP      ENDP


                     /*********************************************
                     *   INTERRUPT AND RESTORE/EXECUTE SECTION   *
                     *********************************************/

131     1        SAVE$REGISTERS:
                                              ; STATEMENT # 131
                     /* THIS ROUTINE IS USED TO SAVE THE STACKED USER'S REGISTERS IN THE
                        MONITOR'S SAVE AREA. */
                          SAVEREGISTERS       PROC NEAR
                05B5  55              PUSH    BP
                05B6  8BEC            MOV     BP,SP
                     PROCEDURE;
132     2               BP = MEMORY$USERSTACK;
                                              ; STATEMENT # 132
                05B8  C41E2A00        LES     BX,MEMORYUSERSTACKPTR
                05BC  268B07          MOV     AX,ES:MEMORYUSERSTACK[BX]
                05BF  A33300          MOV     REGSAV+0AH,AX
133     2               USERSTACK.OFF = USERSTACK.OFF + 4;
                                              ; STATEMENT # 133
                05C2  83062A0004      ADD     USERSTACK,4H
134     2               DO I=0 TO 10;                 /* POP REGISTERS OFF OF STACK */
                                              ; STATEMENT # 134
                05C7  C6065F0000      MOV     I,0H
                         @94:
                05CC  803E5F000A      CMP     I,0AH
                05D1  7724            JA      @95
135     3                     REG$SAV(REG$ORD(I)) = MEMORY$USERSTACK;
                                              ; STATEMENT # 135
                05D3  8A1E5F00        MOV     BL,I
                05D7  B700            MOV     BH,0H
                05D9  2E8A5F67        MOV     BL,CS:REGORD[BX]
                05DD  B700            MOV     BH,0H
                05DF  D1E3            SHL     BX,1
                05E1  C4362A00        LES     SI,MEMORYUSERSTACKPTR
                05E5  268B04          MOV     AX,ES:MEMORYUSERSTACK[SI]
                05E8  89472E          MOV     REGSAV[BX],AX
136     3                     USERSTACK.OFF = USERSTACK.OFF + 2;
                                              ; STATEMENT # 136
```

```
                05EB  83062A0002      ADD     USERSTACK,2H
137   3             END;
                                              ; STATEMENT # 137
                05F0  80065F0001      ADD     I,1H
                05F5  75D5            JNZ     @94
                            @95:
138   2             SS = USERSTACK.SEG;
                                              ; STATEMENT # 138
                05F7  A12C00          MOV     AX,USERSTACK+2H
                05FA  A34200          MOV     REGSAV+14H,AX
139   2             SP = USERSTACK.OFF;
                                              ; STATEMENT # 139
                05FD  A12A00          MOV     AX,USERSTACK
                0600  A33600          MOV     REGSAV+8H,AX
140   2            END;
                                              ; STATEMENT # 140
                0603  5D              POP     BP
                0604  C3              RET
                            SAVEREGISTERS     ENDP

141   1       RESTORE$EXECUTE:
                                              ; STATEMENT # 141
                   /* THIS PROCEDURE RESTORES THE STATE OF THE USER MACHINE AND
                      PASSES CONTROL BACK TO THE USER PROGRAM. IT CONTAINS A
                      MACHINE LANGUAGE SUBROUTINE TO PERFORM THE POPPING OF THE
                      USER REGISTERS AND TO EXECUTE AN 'IRET' TO TRANSFER CONTROL
                      TO THE USER'S PROGRAM. */
                           RESTOREEXECUTE      PROC NEAR
                0605  55              PUSH    BP
                0606  8BEC            MOV     BP,SP
                    PROCEDURE;
142   2                DECLARE RESTORE$EXECUTE$CODE(*) BYTE DATA
                          (08BH,0ECH,           /* MOV BP,SP          */
                           08BH,046H,002H,      /* MOV AX,/BP/.PARM2  */
                           08BH,05EH,004H,      /* MOV BX,/BP/.PARM1  */
                           08EH,0D0H,           /* MOV SS,AX          */
                           08BH,0E3H,           /* MOV SP,BX          */
                           05DH,                /* POP BP             */
                           05FH,                /* POP DI             */
                           05EH,                /* POP SI             */
                           05BH,                /* POP BX             */
                           05AH,                /* POP DX             */
                           059H,                /* POP CX             */
                           058H,                /* POP AX             */
                           01FH,                /* POP DS             */
                           007H,                /* POP ES             */
                           0CFH),               /* IRET               */
                          RESTORE$EXECUTE$CODE$PTR WORD DATA (.RESTORE$EXECUTE$CODE);

143   2                USERSTACK.SEG = SS;
                                              ; STATEMENT # 143
                0608  A14200          MOV     AX,REGSAV+14H
                060B  A32C00          MOV     USERSTACK+2H,AX
144   2                USERSTACK.OFF = SP;
                                              ; STATEMENT # 144
                060E  A13600          MOV     AX,REGSAV+8H
                0611  A32A00          MOV     USERSTACK,AX
```

```
145  2            DO I=0 TO 10;           /* PUSH USER'S REGISTERS ONTO HIS STACK */
                                                ; STATEMENT # 145
         0614  C6065F0000       MOV     I,0H
                    @96:
         0619  803E5F000A       CMP     I,0AH
         061E  7726             JA      @97
146  3               USERSTACK.OFF = USERSTACK.OFF - 2;
                                                ; STATEMENT # 146
         0620  832E2A0002       SUB     USERSTACK,2H
147  3               MEMORY$USERSTACK = REG$SAV(REG$ORD(10-I));
                                                ; STATEMENT # 147
         0625  B30A             MOV     BL,0AH
         0627  2A1E5F00         SUB     BL,I
         062B  B700             MOV     BH,0H
         062D  2E8A5F67         MOV     BL,CS:REGORD[BX]
         0631  B700             MOV     BH,0H
         0633  D1E3             SHL     BX,1
         0635  8B472E           MOV     AX,REGSAV[BX]
         0638  C41E2A00         LES     BX,MEMORYUSERSTACKPTR
         063C  268907           MOV     ES:MEMORYUSERSTACK[BX],AX
148  3           END;
                                                ; STATEMENT # 148
         063F  80065F0001       ADD     I,1H
         0644  75D3             JNZ     @96
                   @97:
149  2           USERSTACK.OFF = USERSTACK.OFF - 2;
                                                ; STATEMENT # 149
         0646  832E2A0002       SUB     USERSTACK,2H
150  2           MEMORY$USERSTACK = BP;
                                                ; STATEMENT # 150
         064B  A13800           MOV     AX,REGSAV+0AH
         064E  C41E2A00         LES     BX,MEMORYUSERSTACKPTR
         0652  268907           MOV     ES:MEMORYUSERSTACK[BX],AX
151  2           CALL RESTORE$EXECUTE$CODE$PTR(USERSTACK.OFF,USERSTACK.SEG);
                                                ; STATEMENT # 151
         0655  FF362A00         PUSH    USERSTACK; 1
         0659  FF362C00         PUSH    USERSTACK+2H; 2
         065D  2EFF160400       CALL    CS:RESTOREEXECUTECODEPTR
152  2           END;
                                                ; STATEMENT # 152
         0662  5D               POP     BP
         0663  C3               RET
                  RESTOREEXECUTE      ENDP

153  1      INTERRUPT1$ENTRY:
                                                ; STATEMENT # 153
           /* THIS PROCEDURE IS CALLED WHEN THE CPU IS INTERRUPTED BY EXECUTING
              AN INSTRUCTION WITH THE TRAP BIT SET (SINGLE STEP). */
         0664  06               PUSH    ES
         0665  1E               PUSH    DS
         0666  2E8E1E9A00       MOV     DS,CS:@@DATA$FRAME
         066B  50               PUSH    AX
         066C  51               PUSH    CX
         066D  52               PUSH    DX
         066E  53               PUSH    BX
         066F  56               PUSH    SI
         0670  57               PUSH    DI
```

```
                0671  E80900          CALL    INTERRUPT1ENTRY
                0674  5F              POP     DI
                0675  5E              POP     SI
                0676  5B              POP     BX
                0677  5A              POP     DX
                0678  59              POP     CX
                0679  58              POP     AX
                067A  1F              POP     DS
                067B  07              POP     ES
                067C  CF              IRET
                            INTERRUPT1ENTRY     PROC NEAR
                067D  55              PUSH    BP
                067E  8BEC            MOV     BP,SP
                       PROCEDURE INTERRUPT 1;
154   2                   USERSTACK.OFF = STACKPTR;          /* SAVE USER STACK INFO */
                                                           ; STATEMENT # 154
                0680  89E0            MOV     AX,SP
                0682  A32A00          MOV     USERSTACK,AX
155   2                   USERSTACK.SEG = STACKBASE;
                                                           ; STATEMENT # 155
                0685  8CD0            MOV     AX,SS
                0687  A32C00          MOV     USERSTACK+2H,AX
156   2                   STACKPTR =  MONITOR$STACKPTR;
                                                           ; STATEMENT # 156
                068A  A11400          MOV     AX,MONITORSTACKPTR
                068D  89C4            MOV     SP,AX
157   2                   STACKBASE = MONITOR$STACKBASE;
                                                           ; STATEMENT # 157
                068F  A11600          MOV     AX,MONITORSTACKBASE
                0692  8ED0            MOV     SS,AX
158   2                   CALL SAVE$REGISTERS;
                                                           ; STATEMENT # 158
                0694  E81EFF          CALL    SAVEREGISTERS
159   2                   FL = FL AND (NOT STEP$TRAP);       /* CLEAR STEP FLAG */
                                                           ; STATEMENT # 159
                0697  81264800FFFE    AND     REGSAV+1AH,0FEFFH
160   2                   IF LAST$COMMAND<>SS$COMMAND THEN
                                                           ; STATEMENT # 160
                069D  803E610003      CMP     LASTCOMMAND,3H
                06A2  7403            JZ      @23
161   2                         CALL RESTORE$EXECUTE;          /* CONTINUE IF NOT SS */
                                                           ; STATEMENT # 161
                06A4  E85EFF          CALL    RESTOREEXECUTE
                            @23:
162   2                   CALL KB$UPDATE$IP;
                                                           ; STATEMENT # 162
                06A7  E8A6FE          CALL    KBUPDATEIP
163   2                   IF CHAR=KBCOM THEN
                                                           ; STATEMENT # 163
                06AA  803E5A0011      CMP     CHAR,11H
                06AF  7515            JNZ     @24
164   2                         DO;
165   3                         IP = CSIP.OFF;
                                                           ; STATEMENT # 165
                06B1  A12600          MOV     AX,CSIP
                06B4  A34600          MOV     REGSAV+18H,AX
166   3                         CS = CSIP.SEG;
```

```
                                                    ; STATEMENT # 166
        06B7  A12800            MOV     AX,CSIP+2H
        06BA  A33E00            MOV     REGSAV+10H,AX
167   3             FL = FL OR STEP$TRAP;          /* SET STEP FLAG */
                                                    ; STATEMENT # 167
        06BD  810E48000001      OR      REGSAV+1AH,100H
168   3             CALL RESTORE$EXECUTE;
                                                    ; STATEMENT # 168
        06C3  E83FFF            CALL    RESTOREEXECUTE
169   3             END;
                     @24:
170   2         IF CHAR<>KBPER THEN  GOTO ERROR;
                                                    ; STATEMENT # 170
        06C6  803E5A0010        CMP     CHAR,10H
        06CB  7403              JZ      $+5H
        06CD  E9ECFA            JMP     ERROR
172   2         GOTO AFTER$COMMAND;
                                                    ; STATEMENT # 172
        06D0  E9D6FA            JMP     AFTERCOMMAND
173   2          END;
                 INTERRUPT1ENTRY      ENDP

174   1       INTERRUPT3$ENTRY:
                                                    ; STATEMENT # 174
           /* THIS PROCEDURE IS CALLED WHEN THE CPU EXECUTES A 'INT 3' INSTRUCTION.
              THE MONITOR INSERTS THIS (OCCH) FOR A BREAKPOINT. ALSO AN EXTERNAL
              INTERRUPT OR A USER SOFTWARE INTERRUPT MAY CAUSE THIS PROCEDURE TO BE
              CALLED. */
        06D3  06                PUSH    ES
        06D4  1E                PUSH    DS
        06D5  2E8E1E9A00        MOV     DS,CS:@@DATA$FRAME
        06DA  50                PUSH    AX
        06DB  51                PUSH    CX
        06DC  52                PUSH    DX
        06DD  53                PUSH    BX
        06DE  56                PUSH    SI
        06DF  57                PUSH    DI
        06E0  E80900            CALL    INTERRUPT3ENTRY
        06E3  5F                POP     DI
        06E4  5E                POP     SI
        06E5  5B                POP     BX
        06E6  5A                POP     DX
        06E7  59                POP     CX
        06E8  58                POP     AX
        06E9  1F                POP     DS
        06EA  07                POP     ES
        06EB  CF                IRET
                 INTERRUPT3ENTRY      PROC NEAR
        06EC  55                PUSH    BP
        06ED  8BEC              MOV     BP,SP
              PROCEDURE INTERRUPT 3;
175   2         USERSTACK.OFF = STACKPTR;       /* SAVE USER STACK INFO */
                                                    ; STATEMENT # 175
        06EF  89E0              MOV     AX,SP
        06F1  A32A00            MOV     USERSTACK,AX
176   2         USERSTACK.SEG = STACKBASE;
                                                    ; STATEMENT # 176
```

```
                06F4  8CD0            MOV     AX,SS
                06F6  A32C00          MOV     USERSTACK+2H,AX
177   2             STACKPTR =  MONITOR$STACKPTR;
                                              ; STATEMENT # 177
                06F9  A11400          MOV     AX,MONITORSTACKPTR
                06FC  89C4            MOV     SP,AX
178   2             STACKBASE = MONITOR$STACKBASE;
                                              ; STATEMENT # 178
                06FE  A11600          MOV     AX,MONITORSTACKBASE
                0701  8ED0            MOV     SS,AX
179   2             CALL SAVE$REGISTERS;
                                              ; STATEMENT # 179
                0703  E8AFFE          CALL    SAVEREGISTERS
180   2             GOTO AFTER$INTERRUPT;
                                              ; STATEMENT # 180
                0706  E9D6FA          JMP     AFTERINTERRUPT
181   2           END;
                          INTERRUPT3ENTRY       ENDP

182   1       INIT$INT$VECTOR:
                                              ; STATEMENT # 182
                  /* THIS ROUTINE INITIALIZES AN INTERRRUPT VECTOR AS FOLLOWS: THE OFFSET
                     FROM THE ADDRESS OF 'INT$ROUTINE' CORRECTED BY THE APPROPRIATE
                     NUMBER OF BYTES FOR THE INTERRUPT PLM PROLOGUE. THE SEGMENT FROM THE
                     CURRENT CS REGISTER IS DETERMINED BY A MACHINE LANGUAGE CODED
                     SUBROUTINE. */
                         INITINTVECTOR      PROC NEAR
                0709  55              PUSH    BP
                070A  8BEC            MOV     BP,SP
                   PROCEDURE(INT$VECTOR$PTR,INT$ROUTINE$OFFSET);
183   2             DECLARE INT$VECTOR$PTR POINTER, INT$ROUTINE$OFFSET WORD,
                        VECTOR BASED INT$VECTOR$PTR STRUCTURE (OFF WORD, SEG WORD),
                        CORRECTION LITERALLY '19H',  /* OFFSET FOR PROLOGUE */
                        INIT$INT$VECTOR$CODE(*) BYTE DATA
                        (055H,                          /* PUSH BP              */
                         08BH,0ECH,                     /* MOV BP,SP            */
                         08CH,0C8H,                     /* MOV AX,CS            */
                         0C4H,05EH,004H,                /* LES BX,/BP/.PARM1    */
                         026H,089H,007H,                /* MOV ES:W/BX/,AX      */
                         05DH,                          /* POP BP               */
                         0C2H,004H,000H),               /* RET 4                */
                        INIT$INT$VECTOR$CODE$PTR WORD DATA (.INIT$INT$VECTOR$CODE);

184   2                 CALL INIT$INT$VECTOR$CODE$PTR(@VECTOR.SEG);    /* SEGMENT PORTION */
                                              ; STATEMENT # 184
                070C  C45E06          LES     BX,[BP].INTVECTORPTR
                070F  268D4702        LEA     AX,ES:VECTOR[BX+2H]
                0713  06              PUSH    ES      ; 1
                0714  50              PUSH    AX      ; 2
                0715  2EFF160600      CALL    CS:INITINTVECTORCODEPTR
185   2                 VECTOR.OFF = INT$ROUTINE$OFFSET - CORRECTION;   /* OFFSET PORTION */
                                              ; STATEMENT # 185
                071A  8B4604          MOV     AX,[BP].INTROUTINEOFFSET
                071D  83E819          SUB     AX,19H
                0720  C45E06          LES     BX,[BP].INTVECTORPTR
                0723  268907          MOV     ES:VECTOR[BX],AX
186   2           END;
```

```
                                                ; STATEMENT # 186
          0726  5D              POP     BP
          0727  C20600          RET     6H
                      INITINTVECTOR     ENDP

              /* ************************************************************************
                 ************************************************************************

                                       COMMAND MODULE
                                       ==============
                 ABSTRACT
                 ========
                 THIS MODULE CONTAINS ALL THE COMMANDS IMPLEMENTED AS INDIVIDUAL PROCEDURES
                 AND CALLED FROM THE OUTER BLOCK OF THE COMMAND DISPATCH LOOP.

                 MODULE ORGANIZATION
                 ===================
                 THIS MODULE CONTAINS THE FOLLOWING SECTIONS:

                    1. COMMANDS SECTION
                        KB$GO               GO
                        KB$SINGLE$STEP      SINGLE STEP
                        KB$EXAM$MEM         SUBSTITUTE MEMORY
                        KB$EXAM$REG         EXAMINE REGISTER
                        KB$MOVE             MOVE
                        KB$INPUT            INPUT PORT
                        KB$OUTPUT           OUTPUT PORT
                      2. COMMAND DISPATCH (OUTER BLOCK, MAIN PROGRAM LOOP)
                        NEXT$COMMAND         DISPATCH
                        ERROR                ERROR ROUTINE
              */

                 /************************
                 *   COMMANDS SECTION   *
                 ************************/

187   1       KB$GO:
                                                ; STATEMENT # 187
                 /* IMPLEMENTS THE 'GO' COMMAND. DISPLAYS IP AND CURRENT INSTRUCTION
                    BYTE AND OPENS CS:IP FOR INPUT AND ONE OPTIONAL BREAKPOINT.
                    BEGINS EXECUTION WHEN A PERIOD IS DEPRESSED AND DISPLAYS 'E' IN
                    THE ADDRESS FIELD. UPON ENCOUNTERING A BREAKPOINT, 'BR' IS
                    DISPLAYED IN THE DATA FIELD. */
                        KBGO      PROC NEAR
          072A  55              PUSH    BP
          072B  8BEC            MOV     BP,SP
                 PROCEDURE;
188   2             CALL KB$UPDATE$IP;                    /* OPTIONAL CHANGE CS:IP */
                                                ; STATEMENT # 188
          072D  E820FE          CALL    KBUPDATEIP
189   2             IF CHAR=KBCOM THEN
                                                ; STATEMENT # 189
          0730  803E5A0011      CMP     CHAR,11H
          0735  754B            JNZ     @26
190   2                 DO;                           /* BREAKPOINT */
191   3                 CALL KB$BLANK$ADDR$FIELD(1);
```

```
                                                ; STATEMENT # 191
        0737  B001          MOV     AL,1H
        0739  50            PUSH    AX      ; 1
        073A  E881FB        CALL    KBBLANKADDRFIELD
192  3              CALL KB$BLANK$DATA$FIELD(0);
                                                ; STATEMENT # 192
        073D  B000          MOV     AL,0H
        073F  50            PUSH    AX      ; 1
        0740  E864FB        CALL    KBBLANKDATAFIELD
193  3              CALL KB$GET$CHAR;
                                                ; STATEMENT # 193
        0743  E855FC        CALL    KBGETCHAR
194  3              CALL KB$GET$ADDR(@BRK1,CSIP.SEG,1);
                                                ; STATEMENT # 194
        0746  8D062200      LEA     AX,BRK1
        074A  1E            PUSH    DS      ; 1
        074B  50            PUSH    AX      ; 2
        074C  FF362800      PUSH    CSIP+2H ; 3
        0750  B001          MOV     AL,1H
        0752  50            PUSH    AX      ; 4
        0753  E8ABFD        CALL    KBGETADDR
195  3              IF CHAR<>KBPER THEN  GOTO ERROR;
                                                ; STATEMENT # 195
        0756  803E5A0010    CMP     CHAR,10H
        075B  7403          JZ      $+5H
        075D  E95CFA        JMP     ERROR
197  3              BRK1$SAVE = MEMORY$BRK1;
                                                ; STATEMENT # 197
        0760  C41E2200      LES     BX,MEMORYBRK1PTR
        0764  268A07        MOV     AL,ES:MEMORYBRK1[BX]
        0767  A25900        MOV     BRK1SAVE,AL
198  3              MEMORY$BRK1 = BREAK$INST;
                                                ; STATEMENT # 198
        076A  26C607CC      MOV     ES:MEMORYBRK1[BX],0CCH
199  3              IF MEMORY$BRK1<>BREAK$INST THEN  GOTO ERROR;
                                                ; STATEMENT # 199
        076E  C41E2200      LES     BX,MEMORYBRK1PTR
        0772  26803FCC      CMP     ES:MEMORYBRK1[BX],0CCH
        0776  7403          JZ      $+5H
        0778  E941FA        JMP     ERROR
201  3              BRK1$FLAG = TRUE;
                                                ; STATEMENT # 201
        077B  C6065800FF    MOV     BRK1FLAG,0FFH
202  3              END;
                                                ; STATEMENT # 202
        0780  EB0A          JMP     @29
                    @26:
                ELSE
203  2              IF CHAR<>KBPER THEN  GOTO ERROR;
                                                ; STATEMENT # 203
        0782  803E5A0010    CMP     CHAR,10H
        0787  7403          JZ      $+5H
        0789  E930FA        JMP     ERROR
                                                ; STATEMENT # 204
                    @29:
                CALL KB$DISPLAY(@KB$EXEC,ADDR$FIELD,0);
                                                ; STATEMENT # 205
```

```
          078C  2E8D062F00      LEA     AX,CS:KBEXEC
          0791  0E              PUSH    CS      ; 1
          0792  50              PUSH    AX      ; 2
          0793  B001            MOV     AL,1H
          0795  50              PUSH    AX      ; 3
          0796  B000            MOV     AL,0H
          0798  50              PUSH    AX      ; 4
          0799  E8B5FA          CALL    KBDISPLAY
206   2           CALL KB$BLANK$DATA$FIELD(0);
                                                ; STATEMENT # 206
          079C  B000            MOV     AL,0H
          079E  50              PUSH    AX      ; 1
          079F  E805FB          CALL    KBBLANKDATAFIELD
207   2           IP = CSIP.OFF;
                                                ; STATEMENT # 207
          07A2  A12600          MOV     AX,CSIP
          07A5  A34600          MOV     REGSAV+18H,AX
208   2           CS = CSIP.SEG;
                                                ; STATEMENT # 208
          07A8  A12800          MOV     AX,CSIP+2H
          07AB  A33E00          MOV     REGSAV+10H,AX
209   2           FL = FL AND (NOT STEP$TRAP);
                                                ; STATEMENT # 209
          07AE  81264800FFFE    AND     REGSAV+1AH,0FEFFH
210   2           CALL RESTORE$EXECUTE;
                                                ; STATEMENT # 210
          07B4  E84EFE          CALL    RESTOREEXECUTE
211   2          END;
                                                ; STATEMENT # 211
          07B7  5D              POP     BP
          07B8  C3              RET
                      KBGO      ENDP

212   1       KB$SINGLE$STEP:
                                                ; STATEMENT # 212
                  /* IMPLEMENTS THE SINGLE STEP COMMAND. DISPLAYS IP AND THE
                     CURRENT INSTRUCTION WORD. OPENS CS:IP FOR INPUT. DEPRESSING
                     COMMA CAUSES THE MONITOR TO SINGLE STEP THE INSTRUCTION,
                     AND PERIOD TERMINATES THE COMMAND. */
                         KBSINGLESTEP      PROC NEAR
          07B9  55              PUSH    BP
          07BA  8BEC            MOV     BP,SP
                 PROCEDURE;
213   2           CALL KB$UPDATE$IP;            /* OPTIONAL CHANGE OF CS:IP */
                                                ; STATEMENT # 213
          07BC  E891FD          CALL    KBUPDATEIP
214   2           IF CHAR<>KBCOM THEN  GOTO ERROR;
                                                ; STATEMENT # 214
          07BF  803E5A0011      CMP     CHAR,11H
          07C4  7403            JZ      $+5H
          07C6  E9F3F9          JMP     ERROR
                    IP = CSIP.OFF;
                                                ; STATEMENT # 216
          07C9  A12600          MOV     AX,CSIP
          07CC  A34600          MOV     REGSAV+18H,AX
217   2           CS = CSIP.SEG;
                                                ; STATEMENT # 217
```

```
          07CF  A12800          MOV     AX,CSIP+2H
          07D2  A33E00          MOV     REGSAV+10H,AX
218   2           FL = FL OR STEP$TRAP;          /* SET TRAP FLAG BIT IN PSW */
                                                 ; STATEMENT # 218
          07D5  810E48000001    OR      REGSAV+1AH,100H
219   2           CALL RESTORE$EXECUTE;
                                                 ; STATEMENT # 219
          07DB  E827FE          CALL    RESTOREEXECUTE
220   2           END;
                                                 ; STATEMENT # 220
          07DE  5D              POP     BP
          07DF  C3              RET
                    KBSINGLESTEP      ENDP

221   1       KB$EXAM$MEM:
                                                 ; STATEMENT # 221
                /* IMPLEMENTS THE EXAMINE MEMORY COMMAND. PROMPTS FOR AN
                   ADDRESS AND THEN DISPLAYS THE BYTE OR WORD AT THAT LOCATION.
                   IT THEN IS OPTIONALLY OPENED FOR INPUT. COMMA INCREMENTS TO
                   THE NEXT LOCATION. PERIOD TERMINATES. */
                       KBEXAMMEM      PROC NEAR
          07E0  55              PUSH    BP
          07E1  8BEC            MOV     BP,SP
                 PROCEDURE;
222   2           DECLARE W WORD;
223   2           CALL KB$BLANK$ADDR$FIELD(1);          /* PROMPT FOR ADDRESS */
                                                 ; STATEMENT # 223
          07E3  B001            MOV     AL,1H
          07E5  50              PUSH    AX      ; 1
          07E6  E8D5FA          CALL    KBBLANKADDRFIELD
224   2           CALL KB$GET$CHAR;
                                                 ; STATEMENT # 224
          07E9  E8AFFB          CALL    KBGETCHAR
225   2           CALL KB$GET$ADDR(@ARG1,CS,1);         /* GET ADDRESS */
                                                 ; STATEMENT # 225
          07EC  8D061A00        LEA     AX,ARG1
          07F0  1E              PUSH    DS      ; 1
          07F1  50              PUSH    AX      ; 2
          07F2  FF363E00        PUSH    REGSAV+10H; 3
          07F6  B001            MOV     AL,1H
          07F8  50              PUSH    AX      ; 4
          07F9  E805FD          CALL    KBGETADDR
226   2           IF CHAR<>KBCOM THEN  GOTO ERROR;
                                                 ; STATEMENT # 226
          07FC  803E5A0011      CMP     CHAR,11H
          0801  7403            JZ      $+5H
          0803  E9B6F9          JMP     ERROR
228   2           DO WHILE TRUE;
                                                 ; STATEMENT # 228
                    @98:
229   3             CALL KB$OUT$WORD(ARG1.OFF,ADDR$FIELD,0,BLANK); /* CLEAR PROMPT */
                                                 ; STATEMENT # 229
          0806  FF361A00        PUSH    ARG1    ; 1
          080A  B001            MOV     AL,1H
          080C  50              PUSH    AX      ; 2
          080D  B100            MOV     CL,0H
          080F  51              PUSH    CX      ; 3
```

```
          0810  50              PUSH    AX      ; 4
          0811  E801FB          CALL    KBOUTWORD
 230   3              IF WORD$MODE THEN
                                                ; STATEMENT # 230
          0814  A06000          MOV     AL,WORDMODE
          0817  D0D8            RCR     AL,1
          0819  7313            JNB     @33
 231   3                 CALL KB$OUT$WORD(MEMORY$WORD$ARG1,DATA$FIELD,1,NOBLANK);
                                                ; STATEMENT # 231
          081B  C41E1A00        LES     BX,MEMORYARG1PTR
          081F  26FF37          PUSH    ES:MEMORYWORDARG1[BX]
          0822  B000            MOV     AL,0H
          0824  50              PUSH    AX      ; 2
          0825  B101            MOV     CL,1H
          0827  51              PUSH    CX      ; 3
          0828  50              PUSH    AX      ; 4
          0829  E8E9FA          CALL    KBOUTWORD
          082C  EB0D            JMP     @34
                          @33:
                        ELSE
 232   3                 CALL KB$OUT$BYTE(MEMORY$ARG1,1);
                                                ; STATEMENT # 232
          082E  C41E1A00        LES     BX,MEMORYARG1PTR
          0832  26FF37          PUSH    ES:MEMORYARG1[BX]
          0835  B001            MOV     AL,1H
          0837  50              PUSH    AX      ; 2
          0838  E89AFA          CALL    KBOUTBYTE
                          @34:
 233   3              CALL KB$GET$CHAR;
                                                ; STATEMENT # 233
          083B  E85DFB          CALL    KBGETCHAR
 234   3              IF CHAR=KBPER THEN RETURN;
                                                ; STATEMENT # 234
          083E  803E5A0010      CMP     CHAR,10H
          0843  7502            JNZ     @35
                                                ; STATEMENT # 235
          0845  5D              POP     BP
          0846  C3              RET
                          @35:
 236   3              IF CHAR<>KBCOM THEN
                                                ; STATEMENT # 236
          0847  803E5A0011      CMP     CHAR,11H
          084C  7503            JNZ     $+5H
          084E  E99100          JMP     @36
 237   3                 IF WORD$MODE THEN
                                                ; STATEMENT # 237
          0851  A06000          MOV     AL,WORDMODE
          0854  D0D8            RCR     AL,1
          0856  7343            JNB     @37
 238   3                    DO;
 239   4                    W = KB$GET$EXPR(DATA$FIELD,1,NOBLANK);
                                                ; STATEMENT # 239
          0858  B000            MOV     AL,0H
          085A  50              PUSH    AX      ; 1
          085B  B101            MOV     CL,1H
          085D  51              PUSH    CX      ; 2
          085E  50              PUSH    AX      ; 3
```

```
                085F  E853FB          CALL    KBGETEXPR
                0862  A34E00          MOV     W,AX
240   4                         IF (CHAR<>KBCOM) AND (CHAR<>KBPER) THEN GOTO ERROR;
                                                ; STATEMENT # 240
                0865  803E5A0011      CMP     CHAR,11H
                086A  B0FF            MOV     AL,0FFH
                086C  7501            JNZ     $+3H
                086E  40              INC     AX
                086F  50              PUSH    AX      ; 1
                0870  803E5A0010      CMP     CHAR,10H
                0875  B0FF            MOV     AL,0FFH
                0877  7501            JNZ     $+3H
                0879  40              INC     AX
                087A  59              POP     CX      ; 1
                087B  22C1            AND     AL,CL
                087D  D0D8            RCR     AL,1
                087F  7303            JNB     $+5H
                0881  E938F9          JMP     ERROR
242   4                         MEMORY$WORD$ARG1 = W;
                                                ; STATEMENT # 242
                0884  A14E00          MOV     AX,W
                0887  C41E1A00        LES     BX,MEMORYARG1PTR
                088B  268907          MOV     ES:MEMORYWORDARG1[BX],AX
243   4                         IF MEMORY$WORD$ARG1<>W THEN  GOTO ERROR;
                                                ; STATEMENT # 243
                088E  C41E1A00        LES     BX,MEMORYARG1PTR
                0892  268B07          MOV     AX,ES:MEMORYWORDARG1[BX]
                0895  3B064E00        CMP     AX,W
                0899  EB42            JMP     @9
245   4                         END;
                                                ; STATEMENT # 245
                        @37:
                          ELSE
246   3                           DO;
247   4                           W = KB$GET$EXPR(DATA$BYTE,1,NOBLANK);
                                                ; STATEMENT # 247
                089B  B0FF            MOV     AL,0FFH
                089D  50              PUSH    AX      ; 1
                089E  B001            MOV     AL,1H
                08A0  50              PUSH    AX      ; 2
                08A1  B000            MOV     AL,0H
                08A3  50              PUSH    AX      ; 3
                08A4  E80EFB          CALL    KBGETEXPR
                08A7  A34E00          MOV     W,AX
248   4                         IF (CHAR<>KBCOM) AND (CHAR<>KBPER) THEN GOTO ERROR;
                                                ; STATEMENT # 248
                08AA  803E5A0011      CMP     CHAR,11H
                08AF  B0FF            MOV     AL,0FFH
                08B1  7501            JNZ     $+3H
                08B3  40              INC     AX
                08B4  50              PUSH    AX      ; 1
                08B5  803E5A0010      CMP     CHAR,10H
                08BA  B0FF            MOV     AL,0FFH
                08BC  7501            JNZ     $+3H
                08BE  40              INC     AX
                08BF  59              POP     CX      ; 1
                08C0  22C1            AND     AL,CL
```

```
                08C2  D0D8            RCR     AL,1
                08C4  7303            JNB     $+5H
                08C6  E9F3F8          JMP     ERROR
250     4                       MEMORY$ARG1 = LOW(W);
                                              ; STATEMENT # 250
                08C9  A14E00          MOV     AX,W
                08CC  C41E1A00        LES     BX,MEMORYARG1PTR
                08D0  268807          MOV     ES:MEMORYARG1[BX],AL
251     4                       IF MEMORY$ARG1<>LOW(W) THEN  GOTO ERROR;
                                              ; STATEMENT # 251
                08D3  A14E00          MOV     AX,W
                08D6  C41E1A00        LES     BX,MEMORYARG1PTR
                08DA  263807          CMP     ES:MEMORYARG1[BX],AL
                            @9:
                08DD  7403            JZ      $+5H
                08DF  E9DAF8          JMP     ERROR
                                              ; STATEMENT # 252
253     4                       END;
                            @36:
254     3                   IF CHAR=KBPER THEN  RETURN;
                                              ; STATEMENT # 254
                08E2  803E5A0010      CMP     CHAR,10H
                08E7  7502            JNZ     @43
                                              ; STATEMENT # 255
                08E9  5D              POP     BP
                08EA  C3              RET
                            @43:
256     3                   IF WORD$MODE THEN
                                              ; STATEMENT # 256
                08EB  A06000          MOV     AL,WORDMODE
                08EE  D0D8            RCR     AL,1
                08F0  7307            JNB     @44
257     3                     ARG1.OFF = ARG1.OFF + 2;
                                              ; STATEMENT # 257
                08F2  83061A0002      ADD     ARG1,2H
                08F7  EB05            JMP     @10
                            @44:
                            ELSE
258     3                     ARG1.OFF = ARG1.OFF + 1;
                                              ; STATEMENT # 258
                08F9  83061A0001      ADD     ARG1,1H
259     3               END;
                                              ; STATEMENT # 259
                            @10:
                08FE  E905FF          JMP     @98
                            @99:
260     2           END;
                                              ; STATEMENT # 260
                0901  5D              POP     BP
                0902  C3              RET
                            KBEXAMMEM       ENDP

261     1       KB$EXAM$REG:
                                              ; STATEMENT # 261
                  /* IMPLEMENTS THE EXAMINE REGISTER COMMAND. PROMPTS FOR A VALID
                     REGISTER KEY AND DISPLAYS THE VALUE OF THAT REGISTER WHICH IS
                     OPTIONALLY OPENED FOR INPUT. COMMA INCREMENTS TO NEXT REGISTER
```

```
                  UNLESS IT IS 'FL' WHICH TERMINATESS AS DOES PERIOD. */
                      KBEXAMREG       PROC NEAR
          0903  55                  PUSH    BP
          0904  8BEC                MOV     BP,SP
                 PROCEDURE;
262   2             DECLARE I BYTE, SAVE WORD;
263   2             CALL KB$BLANK$ADDR$FIELD(1);
                                              ; STATEMENT # 263
          0906  B001                MOV     AL,1H
          0908  50                  PUSH    AX      ; 1
          0909  E8B2F9              CALL    KBBLANKADDRFIELD
264   2             CALL KB$GET$CHAR;
                                              ; STATEMENT # 264
          090C  E88CFA              CALL    KBGETCHAR
265   2             IF CHAR>0DH THEN  GOTO ERROR;            /* INVALID REG KEY */
                                              ; STATEMENT # 265
          090F  803E5A000D          CMP     CHAR,0DH
          0914  7603                JBE     $+5H
          0916  E9A3F8              JMP     ERROR
267   2             I = CHAR;
                                              ; STATEMENT # 267
          0919  A05A00              MOV     AL,CHAR
          091C  A26400              MOV     I,AL
268   2             DO WHILE TRUE;
                                              ; STATEMENT # 268
                    @100:
269   3               DISP(0),DISP(1) = 0;                  /* DISPLAY REG NAME */
                                              ; STATEMENT # 269
          091F  B000                MOV     AL,0H
          0921  A25B00              MOV     DISP,AL
          0924  A25C00              MOV     DISP+1H,AL
270   3               DISP(2) = KB$REG(I*2);
                                              ; STATEMENT # 270
          0927  A06400              MOV     AL,I
          092A  B102                MOV     CL,2H
          092C  F6E1                MUL     CL
          092E  89C3                MOV     BX,AX
          0930  2E8A4F4B            MOV     CL,CS:KBREG[BX]
          0934  880E5D00            MOV     DISP+2H,CL
271   3               DISP(3) = KB$REG(I*2+1);
                                              ; STATEMENT # 271
          0938  2E8A4F4C            MOV     CL,CS:KBREG[BX+1H]
          093C  880E5E00            MOV     DISP+3H,CL
272   3               CALL KB$DISPLAY(@DISP,ADDR$FIELD,0);
                                              ; STATEMENT # 272
          0940  8D065B00            LEA     AX,DISP
          0944  1E                  PUSH    DS      ; 1
          0945  50                  PUSH    AX      ; 2
          0946  B001                MOV     AL,1H
          0948  50                  PUSH    AX      ; 3
          0949  B000                MOV     AL,0H
          094B  50                  PUSH    AX      ; 4
          094C  E802F9              CALL    KBDISPLAY
273   3               CALL KB$OUT$WORD(REG$SAV(I),DATA$FIELD,1,NOBLANK); /* VALUE */
                                              ; STATEMENT # 273
          094F  8A1E6400            MOV     BL,I
          0953  B700                MOV     BH,0H
```

```
                0955  D1E3              SHL     BX,1
                0957  FF772E            PUSH    REGSAV[BX]; 1
                095A  B000              MOV     AL,0H
                095C  50                PUSH    AX      ; 2
                095D  B101              MOV     CL,1H
                095F  51                PUSH    CX      ; 3
                0960  50                PUSH    AX      ; 4
                0961  E8B1F9            CALL    KBOUTWORD
274   3                       CALL KB$GET$CHAR;
                                                        ; STATEMENT # 274
                0964  E834FA            CALL    KBGETCHAR
275   3                       IF CHAR<>KBCOM AND CHAR<>KBPER THEN
                                                        ; STATEMENT # 275
                0967  803E5A0011        CMP     CHAR,11H
                096C  B0FF              MOV     AL,0FFH
                096E  7501              JNZ     $+3H
                0970  40                INC     AX
                0971  50                PUSH    AX      ; 1
                0972  803E5A0010        CMP     CHAR,10H
                0977  B0FF              MOV     AL,0FFH
                0979  7501              JNZ     $+3H
                097B  40                INC     AX
                097C  59                POP     CX      ; 1
                097D  22C1              AND     AL,CL
                097F  D0D8              RCR     AL,1
                0981  733A              JNB     @47
276   3                         DO;
277   4                         SAVE = KB$GET$EXPR(DATA$FIELD,1,NOBLANK); /* UPDATE */
                                                        ; STATEMENT # 277
                0983  B000              MOV     AL,0H
                0985  50                PUSH    AX      ; 1
                0986  B101              MOV     CL,1H
                0988  51                PUSH    CX      ; 2
                0989  50                PUSH    AX      ; 3
                098A  E828FA            CALL    KBGETEXPR
                098D  A35000            MOV     SAVE,AX
278   4                         IF CHAR<>KBCOM AND CHAR<>KBPER THEN GOTO ERROR;
                                                        ; STATEMENT # 278
                0990  803E5A0011        CMP     CHAR,11H
                0995  B0FF              MOV     AL,0FFH
                0997  7501              JNZ     $+3H
                0999  40                INC     AX
                099A  50                PUSH    AX      ; 1
                099B  803E5A0010        CMP     CHAR,10H
                09A0  B0FF              MOV     AL,0FFH
                09A2  7501              JNZ     $+3H
                09A4  40                INC     AX
                09A5  59                POP     CX      ; 1
                09A6  22C1              AND     AL,CL
                09A8  D0D8              RCR     AL,1
                09AA  7303              JNB     $+5H
                09AC  E90DF8            JMP     ERROR
280   4                         REG$SAV(I) = SAVE;
                                                        ; STATEMENT # 280
                09AF  8A1E6400          MOV     BL,I
                09B3  B700              MOV     BH,0H
                09B5  D1E3              SHL     BX,1
```

```
                09B7  A15000          MOV     AX,SAVE
                09BA  89472E          MOV     REGSAV[BX],AX
281   4                 END;
                        @47:
282   3               IF CHAR=KBPER OR I=13 THEN  RETURN;              /* DONE? */
                                                ; STATEMENT # 282
                09BD  803E5A0010      CMP     CHAR,10H
                09C2  B0FF            MOV     AL,0FFH
                09C4  7401            JZ      $+3H
                09C6  40              INC     AX
                09C7  50              PUSH    AX      ; 1
                09C8  803E64000D      CMP     I,0DH
                09CD  B0FF            MOV     AL,0FFH
                09CF  7401            JZ      $+3H
                09D1  40              INC     AX
                09D2  59              POP     CX      ; 1
                09D3  0AC1            OR      AL,CL
                09D5  D0D8            RCR     AL,1
                09D7  7302            JNB     @49
                                                ; STATEMENT # 283
                09D9  5D              POP     BP
                09DA  C3              RET
                        @49:
284   3               I = I + 1;
                                                ; STATEMENT # 284
                09DB  8006640001      ADD     I,1H
285   3           END;
                                                ; STATEMENT # 285
                09E0  E93CFF          JMP     @100
                        @101:
286   2          END;
                                                ; STATEMENT # 286
                09E3  5D              POP     BP
                09E4  C3              RET
                        KBEXAMREG       ENDP

287   1       KB$MOVE:
                                                ; STATEMENT # 287
                    /* IMPLEMENTS THE MOVE COMMAND. PROMPTS FOR 3 ARGUMENTS AND MOVES
                       THE BLOCK OF MEMORY SPECIFIED BY ARG1-ARG2 TO ARG3. IF THERE IS
                       A DIFFERENCE WHEN READ BACK, THEN ERROR. */
                            KBMOVE      PROC NEAR
                09E5  55              PUSH    BP
                09E6  8BEC            MOV     BP,SP
                     PROCEDURE;
288   2              CALL KB$BLANK$ADDR$FIELD(3);              /* FIRST ARGUMENT */
                                                ; STATEMENT # 288
                09E8  B003            MOV     AL,3H
                09EA  50              PUSH    AX      ; 1
                09EB  E8D0F8          CALL    KBBLANKADDRFIELD
289   2              CALL KB$GET$CHAR;
                                                ; STATEMENT # 289
                09EE  E8AAF9          CALL    KBGETCHAR
290   2              CALL KB$GET$ADDR(@ARG1,CS,3);
                                                ; STATEMENT # 290
                09F1  8D061A00        LEA     AX,ARG1
                09F5  1E              PUSH    DS      ; 1
```

```
          09F6  50              PUSH    AX      ; 2
          09F7  FF363E00        PUSH    REGSAV+10H; 3
          09FB  B003            MOV     AL,3H
          09FD  50              PUSH    AX      ; 4
          09FE  E800FB          CALL    KBGETADDR
291   2           IF CHAR<>KBCOM THEN  GOTO ERROR;
                                        ; STATEMENT # 291
          0A01  803E5A0011      CMP     CHAR,11H
          0A06  7403            JZ      $+5H
          0A08  E9B1F7          JMP     ERROR
293   2           CALL KB$BLANK$ADDR$FIELD(2);            /* SECOND ARGUMENT */
                                        ; STATEMENT # 293
          0A0B  B002            MOV     AL,2H
          0A0D  50              PUSH    AX      ; 1
          0A0E  E8ADF8          CALL    KBBLANKADDRFIELD
294   2           CALL KB$GET$CHAR;
                                        ; STATEMENT # 294
          0A11  E887F9          CALL    KBGETCHAR
295   2           END$OFF = KB$GET$EXPR(ADDR$FIELD,2,BLANK);
                                        ; STATEMENT # 295
          0A14  B001            MOV     AL,1H
          0A16  50              PUSH    AX      ; 1
          0A17  B102            MOV     CL,2H
          0A19  51              PUSH    CX      ; 2
          0A1A  50              PUSH    AX      ; 3
          0A1B  E897F9          CALL    KBGETEXPR
          0A1E  A31800          MOV     ENDOFF,AX
296   2           IF END$OFF<ARG1.OFF THEN  GOTO ERROR;
                                        ; STATEMENT # 296
          0A21  A11800          MOV     AX,ENDOFF
          0A24  3B061A00        CMP     AX,ARG1
          0A28  7303            JNB     $+5H
          0A2A  E98FF7          JMP     ERROR
298   2           IF CHAR<>KBCOM THEN  GOTO ERROR;
                                        ; STATEMENT # 298
          0A2D  803E5A0011      CMP     CHAR,11H
          0A32  7403            JZ      $+5H
          0A34  E985F7          JMP     ERROR
300   2           CALL KB$BLANK$ADDR$FIELD(1);             /* THIRD ARGUMENT */
                                        ; STATEMENT # 300
          0A37  B001            MOV     AL,1H
          0A39  50              PUSH    AX      ; 1
          0A3A  E881F8          CALL    KBBLANKADDRFIELD
301   2           CALL KB$GET$CHAR;
                                        ; STATEMENT # 301
          0A3D  E85BF9          CALL    KBGETCHAR
302   2           CALL KB$GET$ADDR(@ARG3,ARG1.SEG,1);
                                        ; STATEMENT # 302
          0A40  8D061E00        LEA     AX,ARG3
          0A44  1E              PUSH    DS      ; 1
          0A45  50              PUSH    AX      ; 2
          0A46  FF361C00        PUSH    ARG1+2H ; 3
          0A4A  B001            MOV     AL,1H
          0A4C  50              PUSH    AX      ; 4
          0A4D  E8B1FA          CALL    KBGETADDR
303   2           IF CHAR<>KBPER THEN  GOTO ERROR;
                                        ; STATEMENT # 303
```

```
          0A50  803E5A0010      CMP     CHAR,10H
          0A55  7403            JZ      $+5H
          0A57  E962F7          JMP     ERROR
305   2       LOOP:
                                        ; STATEMENT # 305
                    LOOP:
                  MEMORY$ARG3 = MEMORY$ARG1;
          0A5A  C41E1A00        LES     BX,MEMORYARG1PTR
          0A5E  268A07          MOV     AL,ES:MEMORYARG1[BX]
          0A61  C41E1E00        LES     BX,MEMORYARG3PTR
          0A65  268807          MOV     ES:MEMORYARG3[BX],AL
306   2             IF MEMORY$ARG3<>MEMORY$ARG1 THEN  GOTO ERROR;
                                        ; STATEMENT # 306
          0A68  C41E1E00        LES     BX,MEMORYARG3PTR
          0A6C  268A07          MOV     AL,ES:MEMORYARG3[BX]
          0A6F  C41E1A00        LES     BX,MEMORYARG1PTR
          0A73  263A07          CMP     AL,ES:MEMORYARG1[BX]
          0A76  7403            JZ      $+5H
          0A78  E941F7          JMP     ERROR
308   2             IF ARG1.OFF = END$OFF THEN RETURN;
                                        ; STATEMENT # 308
          0A7B  A11A00          MOV     AX,ARG1
          0A7E  3B061800        CMP     AX,ENDOFF
          0A82  7502            JNZ     @55
                                        ; STATEMENT # 309
          0A84  5D              POP     BP
          0A85  C3              RET
                    @55:
310   2             ARG1.OFF = ARG1.OFF + 1;
                                        ; STATEMENT # 310
          0A86  83061A0001      ADD     ARG1,1H
311   2             ARG3.OFF = ARG3.OFF + 1;
                                        ; STATEMENT # 311
          0A8B  83061E0001      ADD     ARG3,1H
312   2             GOTO LOOP;
                                        ; STATEMENT # 312
          0A90  EBC8            JMP     LOOP
313   2           END;
                    KBMOVE      ENDP

314   1       KB$INPUT:
                                        ; STATEMENT # 314
                /* PROMPTS FOR A PORT WORD IN THE ADDRESS FIELD. WHEN A COMMA
                   IS ENTERED THE BYTE OR WORD IS DISPLAYED IN THE DATA FIELD.
                   THIS MAY BE REPEATED FOR MULTIPLE READING OF THE PORT. PERIOD
                   TERMINATES THE COMMAND. */
                      KBINPUT       PROC NEAR
          0A92  55              PUSH    BP
          0A93  8BEC            MOV     BP,SP
                  PROCEDURE;
315   2             DECLARE PORT WORD;
316   2             CALL KB$BLANK$ADDR$FIELD(1);                    /* PROMPT FOR PORT */
                                        ; STATEMENT # 316
          0A95  B001            MOV     AL,1H
          0A97  50              PUSH    AX      ; 1
          0A98  E823F8          CALL    KBBLANKADDRFIELD
317   2             CALL KB$GET$CHAR;
```

```
                                                ; STATEMENT # 317
        0A9B  E8FDF8            CALL    KBGETCHAR
318  2             PORT = KB$GET$EXPR(ADDR$FIELD,1,BLANK);      /* GET PORT NUMBER */
                                                ; STATEMENT # 318
        0A9E  B001              MOV     AL,1H
        0AA0  50                PUSH    AX      ; 1
        0AA1  50                PUSH    AX      ; 2
        0AA2  50                PUSH    AX      ; 3
        0AA3  E80FF9            CALL    KBGETEXPR
        0AA6  A35200            MOV     PORT,AX
319  2             CALL KB$OUT$WORD(PORT,ADDR$FIELD,0,BLANK);   /* REMOVE PROMPT */
                                                ; STATEMENT # 319
        0AA9  FF365200          PUSH    PORT    ; 1
        0AAD  B001              MOV     AL,1H
        0AAF  50                PUSH    AX      ; 2
        0AB0  B100              MOV     CL,0H
        0AB2  51                PUSH    CX      ; 3
        0AB3  50                PUSH    AX      ; 4
        0AB4  E85EF8            CALL    KBOUTWORD
320  2      LOOP:
                                                ; STATEMENT # 320
                    LOOP:
        0AB7  803E5A0011        CMP     CHAR,11H
        0ABC  7403              JZ      $+5H
        0ABE  E9FBF6            JMP     ERROR
                IF CHAR<>KBCOM THEN  GOTO ERROR;
322  2          IF WORD$MODE THEN
                                                ; STATEMENT # 322
        0AC1  A06000            MOV     AL,WORDMODE
        0AC4  D0D8              RCR     AL,1
        0AC6  7310              JNB     @57
323  2             CALL KB$OUT$WORD(INWORD(PORT),DATA$FIELD,0,NOBLANK);
                                                ; STATEMENT # 323
        0AC8  8B165200          MOV     DX,PORT
        0ACC  ED                INW     DX
        0ACD  50                PUSH    AX      ; 1
        0ACE  B000              MOV     AL,0H
        0AD0  50                PUSH    AX      ; 2
        0AD1  50                PUSH    AX      ; 3
        0AD2  50                PUSH    AX      ; 4
        0AD3  E83FF8            CALL    KBOUTWORD
        0AD6  EB0C              JMP     @58
                    @57:
                ELSE
324  2             CALL KB$OUT$BYTE(INPUT(PORT),0);
                                                ; STATEMENT # 324
        0AD8  8B165200          MOV     DX,PORT
        0ADC  EC                IN      DX
        0ADD  50                PUSH    AX      ; 1
        0ADE  B000              MOV     AL,0H
        0AE0  50                PUSH    AX      ; 2
        0AE1  E8F1F7            CALL    KBOUTBYTE
                    @58:
325  2          CALL KB$GET$CHAR;
                                                ; STATEMENT # 325
        0AE4  E8B4F8            CALL    KBGETCHAR
326  2          IF CHAR=KBPER THEN RETURN;
```

```
                                                    ; STATEMENT # 326
        0AE7  803E5A0010        CMP     CHAR,10H
        0AEC  75C9              JNZ     LOOP
                                                    ; STATEMENT # 327
        0AEE  5D                POP     BP
        0AEF  C3                RET
328   2             GOTO LOOP;
329   2          END;
                KBINPUT      ENDP

330   1       KB$OUTPUT:
                                                    ; STATEMENT # 330
                /* PROMPTS FOR A PORT WORD IN THE ADDRESS FIELD AND A BYTE OR
                   WORD DATUM IN THE DATA FIELD. THIS DATUM IS OUTPUT A SINGLE TIME
                   TO THE SPECIFIED PORT. */
                    KBOUTPUT      PROC NEAR
        0AF0  55                PUSH    BP
        0AF1  8BEC              MOV     BP,SP
                PROCEDURE;
331   2             DECLARE (DATUM,PORT) WORD;
332   2             CALL KB$BLANK$ADDR$FIELD(1);                    /* PROMPT FOR PORT */
                                                    ; STATEMENT # 332
        0AF3  B001              MOV     AL,1H
        0AF5  50                PUSH    AX      ; 1
        0AF6  E8C5F7            CALL    KBBLANKADDRFIELD
333   2             CALL KB$GET$CHAR;
                                                    ; STATEMENT # 333
        0AF9  E89FF8            CALL    KBGETCHAR
334   2             PORT = KB$GET$EXPR(ADDR$FIELD,1,BLANK);
                                                    ; STATEMENT # 334
        0AFC  B001              MOV     AL,1H
        0AFE  50                PUSH    AX      ; 1
        0AFF  50                PUSH    AX      ; 2
        0B00  50                PUSH    AX      ; 3
        0B01  E8B1F8            CALL    KBGETEXPR
        0B04  A35600            MOV     PORT,AX
335   2             IF CHAR<>KBCOM THEN  GOTO ERROR;
                                                    ; STATEMENT # 335
        0B07  803E5A0011        CMP     CHAR,11H
        0B0C  7403              JZ      $+5H
        0B0E  E9ABF6            JMP     ERROR
337   2             CALL KB$OUT$WORD(PORT,ADDR$FIELD,0,BLANK);   /* REMOVE PROMPT */
                                                    ; STATEMENT # 337
        0B11  FF365600          PUSH    PORT    ; 1
        0B15  B001              MOV     AL,1H
        0B17  50                PUSH    AX      ; 2
        0B18  B100              MOV     CL,0H
        0B1A  51                PUSH    CX      ; 3
        0B1B  50                PUSH    AX      ; 4
        0B1C  E8F6F7            CALL    KBOUTWORD
338   2             CALL KB$BLANK$DATA$FIELD(1);                    /* PROMPT FOR DATUM */
                                                    ; STATEMENT # 338
        0B1F  B001              MOV     AL,1H
        0B21  50                PUSH    AX      ; 1
        0B22  E882F7            CALL    KBBLANKDATAFIELD
339   2             CALL KB$GET$CHAR;
                                                    ; STATEMENT # 339
```

```
                0B25  E873F8          CALL    KBGETCHAR
340   2           LOOP:
                                              ; STATEMENT # 340
                      LOOP:
                0B28  A06000          MOV     AL,WORDMODE
                0B2B  D0D8            RCR     AL,1
                0B2D  7308            JNB     @61
                          IF WORD$MODE THEN
341   2                      DATUM = KB$GET$EXPR(DATA$FIELD,1,NOBLANK); /* GET DATUM WORD */
                                              ; STATEMENT # 341
                0B2F  B000            MOV     AL,0H
                0B31  50              PUSH    AX      ; 1
                0B32  B101            MOV     CL,1H
                0B34  51              PUSH    CX      ; 2
                0B35  EB08            JMP     @18
                            @61:
                          ELSE
342   2                      DATUM = KB$GET$EXPR(DATA$BYTE,1,NOBLANK);  /* GET DATUM BYTE */
                                              ; STATEMENT # 342
                0B37  B0FF            MOV     AL,0FFH
                0B39  50              PUSH    AX      ; 1
                0B3A  B001            MOV     AL,1H
                0B3C  50              PUSH    AX      ; 2
                0B3D  B000            MOV     AL,0H
                            @18:
                0B3F  50              PUSH    AX      ; 3
                0B40  E872F8          CALL    KBGETEXPR
                0B43  A35400          MOV     DATUM,AX
343   2                IF CHAR=KBCOL THEN  GOTO ERROR;
                                              ; STATEMENT # 343
                0B46  803E5A0014      CMP     CHAR,14H
                0B4B  7503            JNZ     $+5H
                0B4D  E96CF6          JMP     ERROR
345   2                IF WORD$MODE THEN
                                              ; STATEMENT # 345
                0B50  A06000          MOV     AL,WORDMODE
                0B53  D0D8            RCR     AL,1
                0B55  730A            JNB     @64
346   2                      OUTWORD(PORT) = DATUM;
                                              ; STATEMENT # 346
                0B57  8B165600        MOV     DX,PORT
                0B5B  A15400          MOV     AX,DATUM
                0B5E  EF              OUTW    DX
                0B5F  EB08            JMP     @65
                            @64:
                          ELSE
347   2                      OUTPUT(PORT) = LOW(DATUM);
                                              ; STATEMENT # 347
                0B61  A15400          MOV     AX,DATUM
                0B64  8B165600        MOV     DX,PORT
                0B68  EE              OUT     DX
                            @65:
348   2                IF CHAR=KBCOM THEN                /* MULTIPLE OUTPUTS */
                                              ; STATEMENT # 348
                0B69  803E5A0011      CMP     CHAR,11H
                0B6E  7510            JNZ     @66
349   2                      DO;
```

```
350   3                 CALL KB$BLANK$DATA$FIELD(1);
                                                  ; STATEMENT # 350
          0B70  B001              MOV     AL,1H
          0B72  50                PUSH    AX      ; 1
          0B73  E831F7            CALL    KBBLANKDATAFIELD
351   3                 CALL KB$GET$CHAR;
                                                  ; STATEMENT # 351
          0B76  E822F8            CALL    KBGETCHAR
352   3                 IF CHAR<>KBPER THEN GOTO LOOP;
                                                  ; STATEMENT # 352
          0B79  803E5A0010        CMP     CHAR,10H
          0B7E  75A8              JNZ     LOOP
                                                  ; STATEMENT # 353
354   3                 END;
                         @66:
355   2           END;
                                                  ; STATEMENT # 355
          0B80  5D                POP     BP
          0B81  C3                RET
                       KBOUTPUT       ENDP


                      /*****************************************
                      *   COMMAND DISPATCH MAIN PROGRAM LOOP   *
                      *****************************************/

356   1               DISABLE;
                                                  ; STATEMENT # 356
          009C  FA                CLI
          009D  2E8E169800        MOV     SS,CS:@@STACK$FRAME
          00A2  BC5000            MOV     SP,@@STACK$OFFSET
          00A5  8BEC              MOV     BP,SP
          00A7  2E8E1E9A00        MOV     DS,CS:@@DATA$FRAME
          00AC  FB                STI
          00AD  FA                CLI
357   1               OUTPUT(KB$STAT$PORT) = KB$INIT$MODE;      /* INIT 8279 */
                                                  ; STATEMENT # 357
          00AE  BAEAFF            MOV     DX,0FFEAH
          00B1  B000              MOV     AL,0H
          00B3  EE                OUT     DX
358   1               OUTPUT(KB$STAT$PORT) = KB$INIT$SCAN;
                                                  ; STATEMENT # 358
          00B4  B039              MOV     AL,39H
          00B6  EE                OUT     DX
359   1               CALL KB$DISPLAY(@KB$SIGNON,ADDR$FIELD,0); /* SIGN ON MESSAGE */
                                                  ; STATEMENT # 359
          00B7  2E8D061F00        LEA     AX,CS:KBSIGNON
          00BC  0E                PUSH    CS      ; 1
          00BD  50                PUSH    AX      ; 2
          00BE  B001              MOV     AL,1H
          00C0  50                PUSH    AX      ; 3
          00C1  B000              MOV     AL,0H
          00C3  50                PUSH    AX      ; 4
          00C4  E88A01            CALL    KBDISPLAY
360   1               CALL KB$DISPLAY(@KB$VERSION,DATA$FIELD,0); /* VERSION NUMBER */
                                                  ; STATEMENT # 360
          00C7  2E8D062300        LEA     AX,CS:KBVERSION
```

```
          00CC  0E              PUSH    CS      ; 1
          00CD  50              PUSH    AX      ; 2
          00CE  B000            MOV     AL,0H
          00D0  50              PUSH    AX      ; 3
          00D1  50              PUSH    AX      ; 4
          00D2  E87C01          CALL    KBDISPLAY

                    /* INITIALIZE USER'S REGISTERS */
361   1             CS,SS,DS,ES,FL,IP = 0;
                                              ; STATEMENT # 361
          00D5  B80000          MOV     AX,0H
          00D8  A33E00          MOV     REGSAV+10H,AX
          00DB  A34200          MOV     REGSAV+14H,AX
          00DE  A34000          MOV     REGSAV+12H,AX
          00E1  A34400          MOV     REGSAV+16H,AX
          00E4  A34800          MOV     REGSAV+1AH,AX
          00E7  A34600          MOV     REGSAV+18H,AX
362   1             SP = USER$INIT$SP;
                                              ; STATEMENT # 362

          00EA  C70636000001    MOV     REGSAV+8H,100H
363   1             CALL INIT$INT$VECTOR(@INT$VECTOR(1),.INTERRUPT1$ENTRY);
                                              ; STATEMENT # 363
          00F0  8D060400        LEA     AX,INTVECTOR+4H
          00F4  1E              PUSH    DS      ; 1
          00F5  50              PUSH    AX      ; 2
          00F6  B87D06          MOV     AX,OFFSET(INTERRUPT1ENTRY)
          00F9  50              PUSH    AX      ; 3
          00FA  E80C06          CALL    INITINTVECTOR
364   1             CALL INIT$INT$VECTOR(@INT$VECTOR(2),.INTERRUPT3$ENTRY);
                                              ; STATEMENT # 364
          00FD  8D060800        LEA     AX,INTVECTOR+8H
          0101  1E              PUSH    DS      ; 1
          0102  50              PUSH    AX      ; 2
          0103  B8EC06          MOV     AX,OFFSET(INTERRUPT3ENTRY)
          0106  50              PUSH    AX      ; 3
          0107  E8FF05          CALL    INITINTVECTOR
365   1             CALL INIT$INT$VECTOR(@INT$VECTOR(3),.INTERRUPT3$ENTRY);
                                              ; STATEMENT # 365
          010A  8D060C00        LEA     AX,INTVECTOR+0CH
          010E  1E              PUSH    DS      ; 1
          010F  50              PUSH    AX      ; 2
          0110  B8EC06          MOV     AX,OFFSET(INTERRUPT3ENTRY)
          0113  50              PUSH    AX      ; 3
          0114  E8F205          CALL    INITINTVECTOR
366   1             BRK1$FLAG = FALSE;
                                              ; STATEMENT # 366
          0117  C606580000      MOV     BRK1FLAG,0H
367   1             MONITOR$STACKBASE = STACKBASE; /* SAVE MONITOR STACK VALUES */
                                              ; STATEMENT # 367
          011C  8CD0            MOV     AX,SS
          011E  A31600          MOV     MONITORSTACKBASE,AX
368   1             MONITOR$STACKPTR = STACKPTR;
                                              ; STATEMENT # 368
          0121  89E0            MOV     AX,SP
          0123  A31400          MOV     MONITORSTACKPTR,AX
369   1             GOTO AFTER$ERROR;
```

```
                                                ; STATEMENT # 369
        0126  EB10          JMP     AFTERERROR

370  1      NEXT$COMMAND:
                                                ; STATEMENT # 370
             /* THIS IS THE PERPETUAL COMMAND LOOP WHICH DISPATCHES TO EACH
                COMMAND WHICH IS A SEPARATE PROCEDURE. */

                  NEXTCOMMAND:
        0128  2E8D061B00    LEA     AX,CS:KBCMNDPMT
        012D  0E            PUSH    CS      ; 1
        012E  50            PUSH    AX      ; 2
        012F  B001          MOV     AL,1H
        0131  50            PUSH    AX      ; 3
        0132  B000          MOV     AL,0H
        0134  50            PUSH    AX      ; 4
        0135  E81901        CALL    KBDISPLAY
             CALL KB$DISPLAY(@KB$CMND$PMT,ADDR$FIELD,0);
371  1      AFTER$ERROR:
                                                ; STATEMENT # 371
                  AFTERERROR:
             CALL KB$GET$CHAR;
        0138  E86002        CALL    KBGETCHAR
372  1           CALL KB$BLANK$ADDR$FIELD(0);
                                                ; STATEMENT # 372
        013B  B000          MOV     AL,0H
        013D  50            PUSH    AX      ; 1
        013E  E87D01        CALL    KBBLANKADDRFIELD
373  1           CALL KB$BLANK$DATA$FIELD(0);
                                                ; STATEMENT # 373
        0141  B000          MOV     AL,0H
        0143  50            PUSH    AX      ; 1
        0144  E86001        CALL    KBBLANKDATAFIELD
374  1           IF (LAST$COMMAND:=CHAR)>09H THEN  GOTO ERROR;
                                                ; STATEMENT # 374
        0147  A05A00        MOV     AL,CHAR
        014A  A26100        MOV     LASTCOMMAND,AL
        014D  3C09          CMP     AL,9H
        014F  776B          JA      ERROR
376  1           WORD$MODE = FALSE;
                                                ; STATEMENT # 376
        0151  C606600000    MOV     WORDMODE,0H
377  1           DO CASE CHAR;
                                                ; STATEMENT # 377
        0156  8A1E5A00      MOV     BL,CHAR
        015A  B700          MOV     BH,0H
        015C  D1E3          SLL     BX,1
        015E  2EFFA79501    JMP     CS:@102[BX]
378  2                CALL KB$EXAM$MEM;
379  2                CALL KB$EXAM$REG;
                                                ; STATEMENT # 379
                   @71:
        0163  E89D07        CALL    KBEXAMREG
        0166  EB41          JMP     AFTERCOMMAND
380  2                CALL KB$GO;
                                                ; STATEMENT # 380
                   @72:
```

```
                0168  E8BF05          CALL    KBGO
                016B  EB3C            JMP     AFTERCOMMAND
381   2                       CALL KB$SINGLE$STEP;
                                                ; STATEMENT # 381
                              @73:
                016D  E84906          CALL    KBSINGLESTEP
                0170  EB37            JMP     AFTERCOMMAND
382   2                       CALL KB$INPUT;
383   2                       CALL KB$OUTPUT;
384   2                       CALL KB$MOVE;
                                                ; STATEMENT # 384
                              @76:
                0172  E87008          CALL    KBMOVE
                0175  EB32            JMP     AFTERCOMMAND
385   2                       DO;  WORD$MODE = TRUE; CALL KB$EXAM$MEM; END;
                              @77:
                                                ; STATEMENT # 386
                0177  C6066000FF      MOV     WORDMODE,0FFH
                                                ; STATEMENT # 387
                              @19:
                017C  E86106          CALL    KBEXAMMEM
                                                ; STATEMENT # 388
                017F  EB28            JMP     AFTERCOMMAND
389   2                       DO;  WORD$MODE = TRUE; CALL KB$INPUT;    END;
                              @78:
                                                ; STATEMENT # 390
                0181  C6066000FF      MOV     WORDMODE,0FFH
                                                ; STATEMENT # 391
                              @21:
                0186  E80909          CALL    KBINPUT
                                                ; STATEMENT # 392
                0189  EB1E            JMP     AFTERCOMMAND
393   2                       DO;  WORD$MODE = TRUE; CALL KB$OUTPUT;   END;
                              @79:
                                                ; STATEMENT # 394
                018B  C6066000FF      MOV     WORDMODE,0FFH
                                                ; STATEMENT # 395
                              @25:
                0190  E85D09          CALL    KBOUTPUT
                                                ; STATEMENT # 396
                0193  EB14            JMP     AFTERCOMMAND
397   2                   END;
                                                ; STATEMENT # 397
                              @102:
                0195  7C01            DW      @19
                0197  6301            DW      @71
                0199  6801            DW      @72
                019B  6D01            DW      @73
                019D  8601            DW      @21
                019F  9001            DW      @25
                01A1  7201            DW      @76
                01A3  7701            DW      @77
                01A5  8101            DW      @78
                01A7  8B01            DW      @79

398   1       AFTER$COMMAND:
                                                ; STATEMENT # 398
```

```
                    AFTERCOMMAND:
          01A9  BC5000          MOV     SP,@@STACK$OFFSET
          01AC  8BEC            MOV     BP,SP
          01AE  2E8E1E9A00      MOV     DS,CS:@@DATA$FRAME
          01B3  B000            MOV     AL,0H
          01B5  50              PUSH    AX      ; 1
          01B6  E8EE00          CALL    KBBLANKDATAFIELD
                    CALL KB$BLANK$DATA$FIELD(0);
399   1             GOTO NEXT$COMMAND;
                                        ; STATEMENT # 399
          01B9  E96CFF          JMP     NEXTCOMMAND

400   1       ERROR:
                                        ; STATEMENT # 400
                  /* THIS ROUTINE HANDLES ALL ERRORS DETECTED BY THE MONITOR AND WILL
                     OUTPUT THE ERROR MESSAGE TO THE DISPLAY AND THEN JUMP TO
                     'AFTER$ERROR' TO GET ANOTHER COMMAND. */

                    ERROR:
          01BC  BC5000          MOV     SP,@@STACK$OFFSET
          01BF  8BEC            MOV     BP,SP
          01C1  2E8E1E9A00      MOV     DS,CS:@@DATA$FRAME
          01C6  2E8D062B00      LEA     AX,CS:KBERRMSG
          01CB  0E              PUSH    CS      ; 1
          01CC  50              PUSH    AX      ; 2
          01CD  B001            MOV     AL,1H
          01CF  50              PUSH    AX      ; 3
          01D0  B000            MOV     AL,0H
          01D2  50              PUSH    AX      ; 4
          01D3  E87B00          CALL    KBDISPLAY
                    CALL KB$DISPLAY(@KB$ERR$MSG,ADDR$FIELD,0);
401   1             CALL KB$BLANK$DATA$FIELD(0);
                                        ; STATEMENT # 401
          01D6  B000            MOV     AL,0H
          01D8  50              PUSH    AX      ; 1
          01D9  E8CB00          CALL    KBBLANKDATAFIELD
402   1             GOTO AFTER$ERROR;
                                        ; STATEMENT # 402
          01DC  E959FF          JMP     AFTERERROR

403   1       AFTER$INTERRUPT:
                                        ; STATEMENT # 403
                  /* THIS ROUTINE CHECKS FOR THE LAST COMMAND WHEN THE MONITOR IS
                     REENTERED VIA THE INTERRUPT VECTOR. */

                    AFTERINTERRUPT:
          01DF  BC5000          MOV     SP,@@STACK$OFFSET
          01E2  8BEC            MOV     BP,SP
          01E4  2E8E1E9A00      MOV     DS,CS:@@DATA$FRAME
          01E9  A05800          MOV     AL,BRK1FLAG
          01EC  D0D8            RCR     AL,1
                    @27:
          01EE  73B9            JNB     AFTERCOMMAND
                    IF BRK1$FLAG THEN
404   1                 DO;
405   2                 MEMORY$BRK1 = BRK1$SAVE;
                                        ; STATEMENT # 405
```

```
         01F0  A05900          MOV     AL,BRK1SAVE
         01F3  C41E2200        LES     BX,MEMORYBRK1PTR
         01F7  268807          MOV     ES:MEMORYBRK1[BX],AL
406   2              BRK1$FLAG = FALSE;
                                        ; STATEMENT # 406
         01FA  C606580000      MOV     BRK1FLAG,0H
407   2              IF ((IP-1) AND 000FH)=(BRK1.OFF AND 000FH) AND
                                        ; STATEMENT # 407
         01FF  A14600          MOV     AX,REGSAV+18H
         0202  83E801          SUB     AX,1H
         0205  50              PUSH    AX      ; 1
         0206  250F00          AND     AX,0FH
         0209  8B0E2200        MOV     CX,BRK1
         020D  51              PUSH    CX      ; 2
         020E  81E10F00        AND     CX,0FH
         0212  3BC1            CMP     AX,CX
         0214  B0FF            MOV     AL,0FFH
         0216  7401            JZ      $+3H
         0218  40              INC     AX
         0219  5A              POP     DX      ; 2
         021A  8756FE          XCHG    DX,[BP]-2H; 1
         021D  B104            MOV     CL,4H
         021F  D3EA            SHR     DX,CL
         0221  03163E00        ADD     DX,REGSAV+10H
         0225  5B              POP     BX      ; 1
         0226  D3EB            SHR     BX,CL
         0228  031E2400        ADD     BX,BRK1+2H
         022C  50              PUSH    AX      ; 1
         022D  3BD3            CMP     DX,BX
         022F  B0FF            MOV     AL,0FFH
         0231  7401            JZ      $+3H
         0233  40              INC     AX
         0234  59              POP     CX      ; 1
         0235  22C1            AND     AL,CL
         0237  D0D8            RCR     AL,1
         0239  73B3            JNB     @27
                        (SHR(IP-1,4)+CS)=(SHR(BRK1.OFF,4)+BRK1.SEG) THEN
408   2                DO;
409   3                IP = IP - 1;
                                        ; STATEMENT # 409
         023B  832E460001      SUB     REGSAV+18H,1H
410   3                CALL KB$DISPLAY(@KB$BRK1,DATA$FIELD,0);
                                        ; STATEMENT # 410
         0240  2E8D063300      LEA     AX,CS:KBBRK1
         0245  0E              PUSH    CS      ; 1
         0246  50              PUSH    AX      ; 2
         0247  B000            MOV     AL,0H
         0249  50              PUSH    AX      ; 3
         024A  50              PUSH    AX      ; 4
         024B  E80300          CALL    KBDISPLAY
411   3                GOTO NEXT$COMMAND;
                                        ; STATEMENT # 411
         024E  E9D7FE          JMP     NEXTCOMMAND
412   3                END;
413   2              END;
414   1            GOTO AFTER$COMMAND;
```

```
415   1           END MONITOR;     /* END OF MODULE */
                EOF
```

CROSS-REFERENCE LISTING

DEFN	ADDR	SIZE	NAME, ATTRIBUTES, AND REFERENCES	
6			ADDRFIELD	LITERALLY 16 32 108 113 119 205 229 272 295 318 319 334 337 359 370 400
398	01A9H		AFTERCOMMAND	LABEL 172 414
371	0138H		AFTERERROR	LABEL 369 402
403	01DFH		AFTERINTERRUPT	LABEL 180
106	0000H	4	ARG	STRUCTURE BASED(PTR) 107 108 112 113
9	001AH	4	ARG1	STRUCTURE AT 225 229 257 258 290 296 302 308 310
9	001EH	4	ARG3	STRUCTURE AT 302 311
34	0006H	1	B	BYTE PARAMETER AUTOMATIC 35 37 38
6			BLANK	LITERALLY 46 108 113 119 229 295 318 319 334 337
62	0004H	1	BLANKING	BYTE PARAMETER AUTOMATIC 63 76 87 96
41	0004H	1	BLANKING	BYTE PARAMETER AUTOMATIC 42 46
13	FHFFF6H	2	BLOCK8089	WORD AT INITIAL ABSOLUTE
13	FHFFF8H	4	BLOCK8089PTR	POINTER AT INITIAL ABSOLUTE
11	FHFFF0H	1	BOOT1	BYTE ARRAY(1) AT INITIAL ABSOLUTE
11	FHFFF1H	2	BOOT2	WORD ARRAY(1) AT INITIAL ABSOLUTE
11	FHFFF3H	2	BOOT3	WORD ARRAY(1) AT INITIAL ABSOLUTE
12	0000H	4	BOOT4	WORD ARRAY(2) DATA
10			BP	LITERALLY 150

```
     6                  BREAKINST               LITERALLY
                                                 198  199

     9  0022H      4    BRK1                    STRUCTURE AT
                                                 194  407

     5  0058H      1    BRK1FLAG                BYTE
                                                 201  366  403  406

     5  0059H      1    BRK1SAVE                BYTE
                                                 197  405

     5  005AH      1    CHAR                    BYTE
                                                  60   67   71   73   80   83   84   97   99
                                                 100  109  114  124  163  170  189  195  203
                                                 214  226  234  236  240  248  254  265  267
                                                 275  278  282  291  298  303  320  326  335
                                                 343  348  352  374  377

     4  0008H     19    COPYRIGHT               BYTE ARRAY(19) DATA

   183                  CORRECTION              LITERALLY
                                                 185

    10                  CS                      LITERALLY
                                                 121  128  225  290  407

     9  0026H      4    CSIP                    STRUCTURE AT
                                                 120  121  128  165  166  194  207  208  216
                                                 217

     6                  DATABYTE                LITERALLY
                                                  74   85   94  247  342

     6                  DATAFIELD               LITERALLY
                                                  28   39  231  239  273  277  323  341  360
                                                 410

   331  0054H      2    DATUM                   WORD
                                                 341  342  346  347

   105  0006H      2    DEFAULTBASE             WORD PARAMETER AUTOMATIC
                                                 106  107

     5  005BH      4    DISP                    BYTE ARRAY(4)
                                                  36   37   38   39   44   49   50   54  269
                                                 270  271  272

    15  0000H      1    DISPLAY                 BYTE BASED(PTR) ARRAY(1)
                                                  20

                       DOUBLE                  BUILTIN
                                                  84

    10                  DS                      LITERALLY
                                                 361
```

```
    5  0018H     2  ENDOFF                WORD
                                           295  296  308

  400  01BCH        ERROR                 LABEL
                                            72   81  101  115  171  196  200  204  215
                                           227  241  244  249  252  266  279  292  297
                                           299  304  307  321  336  344  375

   10                ES                   LITERALLY
                                           361

    6                FALSE                LITERALLY
                                           366  376  406

   14  0006H     1  FIELD                 BYTE PARAMETER AUTOMATIC
                                            15   16

   62  0008H     1  FIELD                 BYTE PARAMETER AUTOMATIC
                                            63   69   74   76   85   87   94   96

   41  0008H     1  FIELD                 BYTE PARAMETER AUTOMATIC
                                            42   54

   10                FL                   LITERALLY
                                           159  167  209  218  361

    6                GOCOMMAND            LITERALLY

  262  0064H     1  I                     BYTE
                                           267  270  271  273  280  282  284

    5  005FH     1  I                     BYTE
                                            19   20   21   43   44   48   49   50   51
                                           134  135  145  147

  182  0709H    33  INITINTVECTOR         PROCEDURE STACK=000EH
                                           363  364  365

  183  0088H    15  INITINTVECTORCODE     BYTE ARRAY(15) DATA
                                           183

  183  0006H     2  INITINTVECTORCODEPTR  WORD DATA
                                           184

                    INPUT                BUILTIN
                                            57   60  324

  153  067DH   111  INTERRUPT1ENTRY       PROCEDURE INTERRUPT(1) STACK=0034H
                                           363

  174  06ECH    54  INTERRUPT3ENTRY       PROCEDURE INTERRUPT(3) STACK=0006H
                                           364  365

  182  0004H     2  INTROUTINEOFFSET      WORD PARAMETER AUTOMATIC
                                           183  185

    2  0000H    20  INTVECTOR             POINTER ARRAY(5)
```

```
                                        363  364  365

  182  0006H     4  INTVECTORPTR        POINTER PARAMETER AUTOMATIC
                                         183  184  185

                    INWORD              BUILTIN
                                         323

   10                 IP                LITERALLY
                                         119  120  215  361  407  409

   30  02BEH    23  KBBLANKADDRFIELD    PROCEDURE STACK=0010H
                                         126  191  223  263  288  293  300  316  332
                                         372

   26  02A7H    23  KBBLANKDATAFIELD    PROCEDURE STACK=0010H
                                         127  192  206  338  350  373  398  401

    7  0037H     4  KBBLANKS            BYTE ARRAY(4) DATA
                                          28   32

    7  0033H     4  KBBRK1              BYTE ARRAY(4) DATA
                                         410

    7  001BH     4  KBCMNDPMT           BYTE ARRAY(4) DATA
                                         370

    8                 KBCOL             LITERALLY
                                          97  109  114  343

    8                 KBCOM             LITERALLY
                                          97  124  163  189  214  226  236  240  248
                                         275  278  291  298  320  335  348

    7                 KBDATAPORT        LITERALLY
                                          23   60

   14  0251H    86  KBDISPLAY           PROCEDURE STACK=000AH
                                          28   32   39   54   69  205  272  359  360
                                         370  400  410

    7  002BH     4  KBERRMSG            BYTE ARRAY(4) DATA
                                         400

  221  07E0H   291  KBEXAMMEM           PROCEDURE STACK=0030H
                                         378  387

  261  0903H   226  KBEXAMREG           PROCEDURE STACK=0024H
                                         379

    7  002FH     4  KBEXEC              BYTE ARRAY(4) DATA
                                         205

  105  0501H    79  KBGETADDR           PROCEDURE STACK=002CH
                                         128  194  225  290  302

   56  039BH    26  KBGETCHAR           PROCEDURE STACK=0002H
```

```
                                                 70   77   88  102  111  123  193  224  233
                                                264  274  289  294  301  317  325  333  339
                                                351  371

    62  03B5H   332  KBGETEXPR                  PROCEDURE WORD STACK=0020H
                                                108  113  239  247  277  295  318  334  341
                                                342

   187  072AH   143  KBGO                       PROCEDURE STACK=0034H
                                                380

     7               KBINITMODE                 LITERALLY
                                                357

     7               KBINITSCAN                 LITERALLY
                                                358

   314  0A92H    94  KBINPUT                    PROCEDURE STACK=0024H
                                                382  391

     7               KBINRDY                    LITERALLY
                                                 57

     8               KBMINUS                    LITERALLY
                                                 99

   287  09E5H   173  KBMOVE                     PROCEDURE STACK=0030H
                                                384

    34  02D5H    64  KBOUTBYTE                  PROCEDURE STACK=0012H
                                                 75   86   95  122  232  324

   330  0AF0H   146  KBOUTPUT                   PROCEDURE STACK=0024H
                                                383  395

    41  0315H   134  KBOUTWORD                  PROCEDURE STACK=0016H
                                                 76   87   96  119  229  231  273  319  323
                                                337

     8               KBPER                      LITERALLY
                                                 97  124  170  195  203  234  240  248  254
                                                275  278  282  303  326  352

     8               KBPLUS                     LITERALLY
                                                 64   91   99

    10  004BH    28  KBREG                      BYTE ARRAY(28) DATA
                                                270  271

     8               KBREGKEY                   LITERALLY
                                                 67

     7  0027H     4  KBREGPMT                   BYTE ARRAY(4) DATA
                                                 69

     7  001FH     4  KBSIGNON                   BYTE ARRAY(4) DATA
                                                359
```

```
 212  07B9H    39  KBSINGLESTEP          PROCEDURE STACK=0034H
                                          381

   7                KBSTATPORT            LITERALLY
                                            17   18   57   59  357  358

 118  0550H   101  KBUPDATEIP            PROCEDURE STACK=0030H
                                          162  188  213

   7  0023H     4  KBVERSION             BYTE ARRAY(4) DATA
                                          360

   5  0061H     1  LASTCOMMAND           BYTE
                                          160  374

   7  003BH    16  LED                   BYTE ARRAY(16) DATA
                                           37   38   44

 305  0A5AH        LOOP                  LABEL
                                          312

 340  0B28H        LOOP                  LABEL
                                          353

 320  0AB7H        LOOP                  LABEL
                                          328

                   LOW                   BUILTIN
                                           75   86   95  250  251  347

   9  0000H     1  MEMORYARG1            BYTE BASED(MEMORYARG1PTR)
                                          232  250  251  305  306

   9  001AH     4  MEMORYARG1PTR         POINTER
                                            9  231  232  242  243  250  251  305  306

   9  0000H     1  MEMORYARG3            BYTE BASED(MEMORYARG3PTR)
                                          305  306

   9  001EH     4  MEMORYARG3PTR         POINTER
                                            9  305  306

   9  0000H     1  MEMORYBRK1            BYTE BASED(MEMORYBRK1PTR)
                                          197  198  199  405

   9  0022H     4  MEMORYBRK1PTR         POINTER
                                            9  197  198  199  405

   9  0000H     1  MEMORYCSIP            BYTE BASED(MEMORYCSIPPTR)
                                          122

   9  0026H     4  MEMORYCSIPPTR         POINTER
                                            9  122

   9  0000H     2  MEMORYUSERSTACK       WORD BASED(MEMORYUSERSTACKPTR)
                                          132  135  147  150
```

```
     9  002AH     4  MEMORYUSERSTACKPTR      POINTER
                                               9  132  135  147  150

     9  0000H     2  MEMORYWORDARG1          WORD BASED(MEMORYARG1PTR)
                                              231  242  243

     1  009CH   437  MONITOR                 PROCEDURE STACK=0050H

     3  0016H     2  MONITORSTACKBASE        WORD
                                              157  178  367

     3  0014H     2  MONITORSTACKPTR         WORD
                                              156  177  368

   370  0128H        NEXTCOMMAND             LABEL
                                              399  411

     6               NOBLANK                 LITERALLY
                                              231  239  247  273  277  323  341  342

     9  0000H     2  OFF                     WORD MEMBER(USERSTACK)
                                              133  136  139  144  146  149  151  154  175

     9  0000H     2  OFF                     WORD MEMBER(CSIP)
                                              120  165  207  216

     9  0000H     2  OFF                     WORD MEMBER(BRK1)
                                              407

     9  0000H     2  OFF                     WORD MEMBER(ARG3)
                                              311

     9  0000H     2  OFF                     WORD MEMBER(ARG1)
                                              229  257  258  296  308  310

   183  0000H     2  OFF                     WORD MEMBER(VECTOR)
                                              185

   106  0000H     2  OFF                     WORD MEMBER(ARG)
                                              108  112  113

    63  0063H     1  OPER                    BYTE
                                               64   91  100

                    OUTPUT                  BUILTIN
                                               17   18   23   59  347  357  358

                    OUTWORD                 BUILTIN
                                              346

   331  0056H     2  PORT                    WORD
                                              334  337  346  347

   315  0052H     2  PORT                    WORD
                                              318  319  323  324

   105  0004H     1  PROMPTS                 BYTE PARAMETER AUTOMATIC
```

```
                                        106  108  113

  62  0006H     1  PROMPTS              BYTE PARAMETER AUTOMATIC
                                          63   69   75   76   86   87   95   96

  41  0006H     1  PROMPTS              BYTE PARAMETER AUTOMATIC
                                          42   54

  34  0004H     1  PROMPTS              BYTE PARAMETER AUTOMATIC
                                          35   39

  30  0004H     1  PROMPTS              BYTE PARAMETER AUTOMATIC
                                          31   32

  26  0004H     1  PROMPTS              BYTE PARAMETER AUTOMATIC
                                          27   28

  14  0004H     1  PROMPTS              BYTE PARAMETER AUTOMATIC
                                          15   21

 105  0008H     4  PTR                  POINTER PARAMETER AUTOMATIC
                                         106  107  108  112  113

  14  0008H     4  PTR                  POINTER PARAMETER AUTOMATIC
                                          15   20

  10  0067H    11  REGORD               BYTE ARRAY(11) DATA
                                         135  147

  10  002EH    28  REGSAV               WORD ARRAY(14)
                                          73  119  120  121  128  132  135  138  139
                                         143  144  147  150  159  165  166  167  207
                                         208  209  216  217  218  225  273  280  290
                                         361  362  407  409

 141  0605H    95  RESTOREEXECUTE       PROCEDURE STACK=0008H
                                         161  168  210  219

 142  0072H    22  RESTOREEXECUTECODE   BYTE ARRAY(22) DATA
                                         142

 142  0004H     2  RESTOREEXECUTECODEPTR WORD DATA
                                         151

 262  0050H     2  SAVE                 WORD
                                         277  280

  63  004AH     2  SAVE                 WORD
                                          73   75   76   82   84   86   87   92   93

 131  05B5H    80  SAVEREGISTERS        PROCEDURE STACK=0002H
                                         158  179

   9  0002H     2  SEG                  WORD MEMBER(USERSTACK)
                                         138  143  151  155  176

   9  0002H     2  SEG                  WORD MEMBER(CSIP)
```

```
                                                121  166  194  208  217

     9  0002H     2  SEG                        WORD MEMBER(BRK1)
                                                 407

     9  0002H     2  SEG                        WORD MEMBER(ARG3)

     9  0002H     2  SEG                        WORD MEMBER(ARG1)
                                                 302

   183  0002H     2  SEG                        WORD MEMBER(VECTOR)
                                                 184

   106  0002H     2  SEG                        WORD MEMBER(ARG)
                                                 107  112

                    SHL                         BUILTIN
                                                  84

                    SHR                         BUILTIN
                                                  37   44  407

    10              SP                          LITERALLY
                                                 144

    10              SS                          LITERALLY
                                                 143  361

     6              SSCOMMAND                   LITERALLY
                                                 160

                    STACKBASE                   BUILTIN
                                                 155  157  176  178  367

                    STACKPTR                    BUILTIN
                                                 154  156  175  177  368

    11              STARTADDR                   LITERALLY
                                                  11   12

     6              STEPTRAP                    LITERALLY
                                                 159  167  209  218

    15  0062H     1  T                          BYTE
                                                  20   22   23

     6              TRUE                        LITERALLY
                                                  66  201  228  268  386  390  394

     6              USERINITSP                  LITERALLY
                                                 362

     9  002AH     4  USERSTACK                  STRUCTURE AT
                                                 133  136  138  139  143  144  146  149  151
                                                 154  155  175  176

   183  0000H     4  VECTOR                     STRUCTURE BASED(INTVECTORPTR)
```

```
                                                    184  185

    222  004EH     2  W                             WORD
                                                    239  242  243  247  250  251

     63  004CH     2  W                             WORD
                                                     65   92   93   95   96   98

     41  000AH     2  W                             WORD PARAMETER AUTOMATIC
                                                     42   44

      5  0060H     1  WORDMODE                      BYTE
                                                    230  237  256  322  340  345  376  386  390
                                                    394

MODULE INFORMATION:

     CODE AREA SIZE     = 0B82H   2946D
     CONSTANT AREA SIZE = 0000H      0D
     VARIABLE AREA SIZE = 0065H    101D
     MAXIMUM STACK SIZE = 0050H     80D
     865 LINES READ
     0 PROGRAM ERROR(S)

END OF PL/M-86 COMPILATION
```

SDK-86 SERIAL MONITOR
PL/M-86 SOURCE LISTING


```
ISIS-II PL/M-86 V1.1 COMPILATION OF MODULE MONITOR
OBJECT MODULE PLACED IN :F1:SERIAL.OBJ
COMPILER INVOKED BY:  PLM86 :F1:SERIAL.PLM LARGE OPTIMIZE(2) PAGEWIDTH(95)


$TITLE('SDK86 SERIAL MONITOR')
$NOINTVECTOR

 /* ***********************************************************************
    ***********************************************************************

         SDK-86 MONITOR, V1.2
         06 AUGUST 1979

         (C) 1978 INTEL CORPORATION. ALL RIGHTS RESERVED. NO PART
    OF THIS PROGRAM OR PUBLICATION MAY BE REPRODUCED, TRANSMITTED,
    TRANSCRIBED, STORED IN A RETRIEVAL SYSTEM, OR TRANSLATED INTO ANY
    LANGUAGE, IN ANY FORM OR BY ANY MEANS, ELECTRONIC, MECHANICAL,
    MAGNETIC, OPTICAL, CHEMICAL, MANUAL OR OTHERWISE, WITHOUT THE PRIOR
    WRITTEN PERMISSION OF INTEL CORPORATION, 3065 BOWERS AVENUE, SANTA
    CLARA, CALIFORNIA, 95051.

    ***********************************************************************
    ***********************************************************************

    ABSTRACT
    ========
    THIS PROGRAM IS THE ROM BASED SERIAL MONITOR FOR THE SDK-86.  IT
    PROVIDES THE USER WITH A MODERATE LEVEL OF CAPABLILITY TO
    EXAMINE/MODIFY MEMORY AND REGISTERS, CONTROL PROGRAM EXECUTION,
    AND LOAD/SAVE PROGRAMS.

    ENVIRONMENT
    ===========
    THE SDK SERIAL MONITOR COMMUNICATES WITH THE USER VIA AN INTERACTIVE
    TERMINAL (TTY,CRT) ATTACHED TO THE SERIAL PORT.

    PROGRAM ORGANIZATION
    ====================
    THE PROGRAM IS DIVIDED INTO 1 DATA AND 2 CODE MODULES:
       1. DATA DECLARATION MODULE.    GLOBAL DATA DECLARATIONS.
       2. COMMON ROUTINES.            LOWER LEVEL PROCEDURES
       3. COMMAND MODULE.             INDIVIDUAL COMMANDS AND OUTER BLOCK

    CALLING GRAPH
    =============
       >>COMMAND DISPATCH MODULE (OUTER BLOCK)
            INDIVIDUAL COMMAND PROCEDURES
               COMMON ROUTINES

    GLOBAL DATA STRUCTURES
    ======================
    THE MONITOR MAINTAINS THE USER'S MACHINE STATE (REGISTERS) IN A
    WORD ARRAY. THE REGISTERS ARE SAVED FROM THE USER'S STACK
    AS PUSHED BY PLM86 INTERRUPT PROCEDURE.
    POINTERS TO THE SDK-86 2**20 ADDRESS SPACE ARE IMPLEMENTED WITH
```

```
                POINTER STRUCTURES ALLOCATED AS 2 WORD STRUCTURES.

              */

  1             MONITOR:DO;                     /* BEGINNING OF MODULE */

              /* ********************************************************************
                 ********************************************************************

                                 GLOBAL DATA DECLARATIONS MODULE
                                 ===============================
                 ABSTRACT
                 ========
                 THIS MODULE CONTAINS ALL THE GLOBAL DATA DECLARATIONS AND
                 LITERALS (EQUATES).

                 MODULE ORGAINIZATION
                 ====================
                 THE MODULE IS DIVIDED INTO 5 SECTIONS:
                    1. UTILITY SECTION              GLOBAL FLAGS,VARIBLES,EQUATES
                    2. I/O SECTION                  I/O PORTS,MASKS,AND SPECIAL CHARS
                    3. MEMORY ARGUMENTS SECTIONS    STRUCTURES FOR POINTERS
                    4. REGISTER SECTION             USER REGISTER SAVE AREA
                    5. BOOT AND 8089 SECTION        BOOTSTRAP AND 8089 DESCRIPTOR
              */

                 /***********************
                 *   UTILITY SECTION   *
                 ***********************/

  2   1         DECLARE
                       INT$VECTOR(5)      POINTER;            /* INTERRUPT VECTORS */
  3   1         DECLARE
                       MONITOR$STACKPTR   WORD,
                       MONITOR$STACKBASE  WORD;
  4   1         DECLARE
                       COPYRIGHT(*) BYTE DATA ('(C) 1978 INTEL CORP');
  5   1         DECLARE
                       BRK1$FLAG       BYTE,               /* TRUE IF BREAK SET */
                       BRK1$SAVE       BYTE,               /* INST BREAK SAVE */
                       CHAR            BYTE,               /* ONE CHAR LOOK AHEAD */
                       CHECK$SUM       BYTE,               /* PAPER TAPE CHECKSUM */
                       I               BYTE,               /* INDEX */
                       END$OFF         WORD,                /* END OFFSET ADDRESS */
                       WORD$MODE       BYTE,               /* WORD MODE FLAG */
                       LAST$COMMAND    BYTE;               /* LAST COMMAND SAVE */
  6   1         DECLARE
                       TRUE            LITERALLY '0FFH',
                       FALSE           LITERALLY '000H',
                       BREAK$INST      LITERALLY '0CCH',   /* BREAKPOINT INST */
                       STEP$TRAP       LITERALLY '0100H',  /* SS TRAP FLAG MASK */
                       USER$INIT$SP    LITERALLY '100H',   /* USER STACK INITIAL */
                       GO$COMMAND      LITERALLY '2',      /* GO COMMAND CODE */
                       SS$COMMAND      LITERALLY '3',      /* SINGLE STEP CODE */
                       STANDARD$LEN    LITERALLY '16',     /* PAPER TAPE DATA REC LEN */
                       MAX$DELAY       LITERALLY '10000';  /* DELAY FOR PAPER TAPE */
  7   1         DECLARE
```

```
                KB$SIGNON(*)      BYTE DATA  (5BH,86H,00H,00H,7DH,7FH,00H,00H),
                                             /* " 2  1.          6   8        " */
                SIO$BREAK$MSG(*) BYTE DATA  ('BR ',0),
                SIO$SIGNON(*)     BYTE DATA  (0DH,0AH,'SDK-86 MONITOR, V1.2',0),
                ASCII(*)         BYTE DATA  ('0123456789ABCDEF'),
                SIO$CMND(*)      BYTE DATA  ('SXGNMDIORW');

             /*********************************
             *   I/O DECLARATIONS SECTION    *
             *********************************/

 8   1       DECLARE
                SIO$STAT$PORT    LITERALLY '0FFF2H',     /* 8251 USART */
                SIO$DATA$PORT    LITERALLY '0FFF0H',     /* STATUS INPUT PORT */
                SIO$8251$RESET   LITERALLY   '065H',     /* DATA INPUT PORT */
                SIO$8251$MODE    LITERALLY   '0CFH',     /* RESET COMMAND */
                SIO$8251$CMND    LITERALLY   '025H',     /* ASYNC MODE */
                SIO$8251$DTR$ON  LITERALLY   '027H',     /* DTR OFF */
                SIO$DSRDY        LITERALLY    '80H',     /* SAME AS CMND BUT DTR ON */
                SIO$RXRDY        LITERALLY    '02H',     /* DATA SET READY */
                SIO$TXRDY        LITERALLY    '01H';     /* RECEIVE DATA READY */
 9   1       DECLARE                                     /* XMIT BUFFER EMPTY */
                KB$STAT$PORT     LITERALLY 'OFFEAH',     /* STATUS/COMMAND PORT */
                KB$DATA$PORT     LITERALLY 'OFFE8H',     /* DATA PORT      */
                KB$INIT$MODE     LITERALLY '00H',        /* 8 8-BIT, LEFT ENTRY,
                                                           ENCODE, 2 KEY LOCKOUT */
                KB$INIT$SCAN     LITERALLY '39H';        /* 10MS SCAN RATE       */
10   1       DECLARE
                ASCR             LITERALLY  '0DH',       /* CARRIAGE RETURN */
                ASLF             LITERALLY  '0AH',       /* LINE FEED       */
                ASBL             LITERALLY  '20H';       /* BLANK OR SPACE  */

             /*********************************
             *   MEMORY ARGUMENT SECTION     *
             *********************************/

11   1       DECLARE
                MEMORY$ARG1$PTR  POINTER,                /* ARGUMENT 1 */
                ARG1 STRUCTURE (OFF WORD, SEG WORD)
                   AT (@MEMORY$ARG1$PTR),
                MEMORY$ARG1 BASED MEMORY$ARG1$PTR BYTE,
                MEMORY$WORD$ARG1 BASED MEMORY$ARG1$PTR WORD,

                MEMORY$ARG3$PTR POINTER,                 /* ARGUMENT 3 */
                ARG3 STRUCTURE (OFF WORD, SEG WORD)
                   AT (@MEMORY$ARG3$PTR),
                MEMORY$ARG3 BASED MEMORY$ARG3$PTR BYTE,

                MEMORY$BRK1$PTR POINTER,                 /* BREAKPOINT 1 */
                BRK1 STRUCTURE (OFF WORD, SEG WORD)
                   AT (@MEMORY$BRK1$PTR),
                MEMORY$BRK1 BASED MEMORY$BRK1$PTR BYTE,

                MEMORY$CSIP$PTR POINTER,                 /* CS & IP WORD */
                CSIP STRUCTURE (OFF WORD, SEG WORD)
                   AT (@MEMORY$CSIP$PTR),
                MEMORY$CSIP BASED MEMORY$CSIP$PTR BYTE,
```

```
                  MEMORY$USERSTACK$PTR POINTER,
                  USERSTACK STRUCTURE (OFF WORD, SEG WORD)
                     AT (@MEMORY$USERSTACK$PTR),
                  MEMORY$USERSTACK BASED MEMORY$USERSTACK$PTR WORD;

               /************************
               *   REGISTER SECTION   *
               ************************/

12   1      DECLARE
                  REG(*)          BYTE DATA              /* REGISTER NAMES */
                     ('AXBXCXDXSPBPSIDICSDSSSESIPFL'),
                  REG$INDEX       WORD,
                  REG$SAV(14)     WORD,                  /* USER'S SAVED REGS */
                  REG$ORD(*)      BYTE DATA
                     (7,6,1,3,2,0,9,11,12,8,13),
                  SP              LITERALLY 'REG$SAV( 4)',
                  BP              LITERALLY 'REG$SAV( 5)',
                  CS              LITERALLY 'REG$SAV( 8)',
                  DS              LITERALLY 'REG$SAV( 9)',
                  SS              LITERALLY 'REG$SAV(10)',
                  ES              LITERALLY 'REG$SAV(11)',
                  IP              LITERALLY 'REG$SAV(12)',
                  FL              LITERALLY 'REG$SAV(13)';


               /************************************
               *   BOOTSTRAP JUMP AND 8089 VECTOR   *
               ************************************/

               /* THIS BOOT CONSISTS OF A LONG JUMP TO THE BEGINNING OF THE MONITOR
                  AT FF00+XXXX WHERE XXXX IS THE STARTING ADDRESS OFFSET (IP) AND
                  MUST BE DETERMINED AFTER EACH COMPILE. */
13   1      DECLARE
                  START$ADDR LITERALLY '00A8H',              /* STARTING ADDRESS */
                  BOOT1(*)   BYTE AT (0FFFF0H) DATA (0EAH),  /* LONG JUMP OPCODE */
                  BOOT2(*)   WORD AT (0FFFF1H) DATA (START$ADDR),
                  BOOT3(*)   WORD AT (0FFFF3H) DATA (0FF00H); /* SEGMENT ADDRESS! */

               /* THIS TWO-WORD DATA IS A JUMP TO THE STARTING ADDRESS AND IS LOCATED
                  AT THE FIRST LOCATION OF ROM (NO OTHER DATA OR CONSTANT DECLARATIONS
                  MAY PRECEDE IT). THE JUMP IS ACTUALLY TO (START-ADDR)-4 SINCE THE
                  INSTRUCTION IS A RELATIVE JUMP OF LENGTH 3. */
14   1      DECLARE
                  BOOT4(*)   WORD DATA (0E990H,START$ADDR-4);  /* NOP,JMP START-ADDR */

               /* THIS BLOCK OF ROM AT FFFF6-FFFFA IS INITIALIZED FOR THE 8089
                  DEVICE AND POINTS TO A BLOCK OF RAM AT 0100H. */
15   1      DECLARE
                  BLOCK$8089      WORD    AT (0FFFF6H) DATA (00001H),
                  BLOCK$8089$PTR  POINTER AT (0FFFF8H) DATA (00100H);

               /*********************************
               *   FILL DECLARATIONS SECTION   *
               *********************************/
```

```
                /* THESE DECLARATIONS DEFINE FILL CHARACTERS FOR ALL UNUSED
                   LOCATIONS IN THE ROM ADDRESS SPACE. ANY MODIFICATION TO THE CODE
                   THAT CHANGES THE USED LOCATIONS IN THE ROM REQUIRES
                   MODIFICATION OF THIS SECTION. */

 16   1         DECLARE

                    F        LITERALLY 'OFFH',
                    FILL1(*) BYTE AT (OFFFDBH) DATA (F,F,F,F,F),
                    FILL2(*) BYTE AT (OFFFE0H) DATA (F,F,F,F,F,F,F,F,F,F,F,F,F,F,F,F),
                    FILL3(*) BYTE AT (OFFFF5H) DATA (F),
                    FILL4(*) BYTE AT (OFFFFCH) DATA (F,F,F,F);


                /* ************************************************************************
                   ************************************************************************

                                   COMMON PROCEDURES
                                   =================
                   ABSTRACT
                   ========
                   THIS MODULE CONTAINS THOSE LOWER LEVEL PROCEDURES CALLED BY HIGHER
                   LEVEL ROUTINES.

                   MODULE ORGANIZATION
                   ===================
                   THIS MODULE IS DIVIDED INTO 4 SECTIONS AS FOLLOWS:
                      1. BASIC I/O SECTION
                          SIO$CHAR$RDY          INPUT CHARACTER READY
                          SIO$CHECK$CONTROL$CHAR   CHECK FOR CONTROL CHARACTER
                          SIO$OUT$CHAR          OUTPUT CHARACTER
                          SIO$GET$CHAR          INPUT A CHARACTER
                          SIO$OUT$BYTE          OUTPUT A BYTE IN HEX
                          SIO$OUT$WORD          OUTPUT A WORD IN HEX
                          SIO$OUT$BLANK         OUTPUT A SINGLE BLANK
                          SIO$OUT$STRING        OUTPUT A STRING
                          SIO$OUT$HEADER        OUTPUT A PAPER TAPE HEADER
                      2. UTILITY ROUTINES SECTION
                          SIO$VALID$HEX         TEST FOR VALID HEX CHAR
                          SIO$HEX               CONVERT TO HEX FROM ASCII
                          SIO$VALID$REG$FIRST   TEST FOR VALID REGISTER FIRST CHAR
                          SIO$VALID$REG         TEST FOR VALID REGISTER NAME
                          SIO$CRLF              OUTPUTS A CR AND LF
                          SIO$TEST$WORD$MODE    TEST FOR A 'W' IN COMMAND
                          SIO$SCAN$BLANK        SCANS FOR OPTIONAL BLANK
                      3. ARGUMENT EXPRESSION EVALUATION SECTION
                          SIO$GET$WORD          GET AN WORD EXPRESSION
                          SIO$GET$ADDR          GET AN ADDRESS EXPRESSION
                          SIO$UPDATE$IP         OPTIONAL UPDATE OF CS:IP
                      4. PAPER TAPE READ SECTION
                          SIO$READ$CHAR         READ CHAR FROM TTY READER
                          SIO$READ$BYTE         READ A BYTE
                          SIO$READ$WORD         READ A WORD
                      5. INTERRUPT AND RESTORE/EXECUTE ROUTINES
                          SAVE$REGISTERS        SAVES USER REGISTERS
                          RESTORE$EXECUTE       RESTORE MACHINE STATE AND EXEC
                          INTERRUPT1$ENTRY      INTERRUPT ROUTINE FOR SINGLE STEP
```

```
                        INTERRUPT3$ENTRY     INTERRUPT ROUTINE FOR GO
                        INIT$INT$VECTOR      INITIALIZES INTERRRUPT VECTORS
                  */

                     /************************
                     *   BASIC I/O SECTION   *
                     ************************/

 17   1          SIO$CHAR$RDY:
                    /* TESTS FOR INPUT CHARACTER PENDING BY READING THE STATUS PORT
                       AND MASKING WITH SIO$RXRDY(READ DATA READY).  RETURNS TRUE IF
                       NOT EMPTY(CHAR PENDING) AND FALSE IF NO CHAR PENDING */
                    PROCEDURE BYTE;
 18   2                 IF (INPUT(SIO$STAT$PORT) AND SIO$RXRDY)=0 THEN RETURN FALSE;
 20   2                 RETURN TRUE;
 21   2              END;

 22   1          SIO$CHECK$CONTROL$CHAR:
                    /* THIS ROUTINE CHECKS IF A CONTROL CHARACTER HAS BEEN INPUT TO
                       THE SERIAL PORT.  AFTER A CONTROL-S IT WAITS FOR A CONTROL-Q BEFORE
                       RETURNING TO THE CALLER.  A CONTROL-C CAUSES A JUMP TO THE ERROR
                       ROUTINE. */
                    PROCEDURE;
 23   2                 CHAR = INPUT(SIO$DATA$PORT) AND 07FH;
 24   2                 IF CHAR=13H THEN                /* CONTROL-S ? */
 25   2                    DO WHILE CHAR<>11H;          /* CONTROL-Q   */
 26   3                       IF SIO$CHAR$RDY THEN
 27   3                          DO;
 28   4                          CHAR = INPUT(SIO$DATA$PORT) AND 07FH;
 29   4                          IF CHAR=03H THEN GOTO ERROR;
 31   4                          END;
 32   3                    END;
 33   2                 ELSE IF CHAR = 03H THEN GOTO ERROR;
                    END SIO$CHECK$CONTROL$CHAR;

 36   1          SIO$OUT$CHAR:
                    /* THIS ROUTINE OUTPUTS THE INPUT PARAMETER TO THE USART OUTPUT
                       PORT WHEN USART IS READY FOR OUTPUT (XMIT BUFFER EMPTY). */
                    PROCEDURE(C);
 37   2                 DECLARE C BYTE;
 38   2                 IF SIO$CHAR$RDY THEN CALL SIO$CHECK$CONTROL$CHAR;
 40   2                 DO WHILE (INPUT(SIO$STAT$PORT) AND SIO$TXRDY)=0;END;
 42   2                 OUTPUT(SIO$DATA$PORT) = C;
 43   2              END;

 44   1          SIO$GET$CHAR:
                    /* THIS ROUTINE INPUTS A CHARACTER FROM THE INPUT PORT AND RETURNS
                       WITH IT IN THE GLOBAL 'CHAR'. THE CHARACTER IS ECHOED TO THE
                       OUTPUT PORT IF PRINTABLE. */
                    PROCEDURE;
 45   2                 DO WHILE (INPUT(SIO$STAT$PORT) AND SIO$RXRDY)=0;END;
 47   2                 CHAR = INPUT(SIO$DATA$PORT) AND 07FH;
 48   2                 IF CHAR>=ASBL THEN CALL SIO$OUT$CHAR(CHAR);
 50   2              END;

 51   1          SIO$OUT$BYTE:
                    /* THIS ROUTINE OUTPUTS THE SINGLE INPUT PARAMETER TO THE USART
```

```
                        IN ASCII HEXADECIMAL FORMAT. */
                     PROCEDURE(B);
  52    2                DECLARE B BYTE;
  53    2                CALL SIO$OUT$CHAR(ASCII(SHR(B,4) AND 0FH));
  54    2                CALL SIO$OUT$CHAR(ASCII(B AND 0FH));
  55    2                CHECK$SUM = CHECK$SUM - B;
  56    2             END;

  57    1          SIO$OUT$WORD:
                     /* THIS ROUTINE OUTPUTS THE INPUT PARAMETER AS 4 ASCII HEXADECIMAL
                        CHARACTERS TO THE USART OUTPUT PORT. */
                     PROCEDURE(W);
  58    2                DECLARE W WORD;
  59    2                CALL SIO$OUT$BYTE(HIGH(W));
  60    2                CALL SIO$OUT$BYTE(LOW(W));
  61    2             END;

  62    1          SIO$OUT$BLANK:
                     /* THIS ROUTINE OUTPUTS ONE BLANK. */
                     PROCEDURE;
  63    2                CALL SIO$OUT$CHAR(ASBL);
  64    2             END;

  65    1          SIO$OUT$STRING:
                     /* OUTPUTS A STRING POINTED TO BY THE FIRST PARM. */
                     PROCEDURE(PTR);
  66    2                DECLARE PTR POINTER, STR BASED PTR (1) BYTE;
  67    2                I = 0;
  68    2                DO WHILE STR(I)<>0;
  69    3                   CALL SIO$OUT$CHAR(STR(I));
  70    3                   I = I + 1;
  71    3                END;
  72    2             END;

  73    1          SIO$OUT$HEADER:
                     /* THIS ROUTINE OUTPUTS THE PAPER TAPE HEADER CONSISTING OF ':'
                        FOLLOWED BY THE RECORD LENGTH, LOAD ADDRESS, AND THE RECORD TYPE.
                        IT INITIALIZES THE CHECKSUM TO ZERO. */
                     PROCEDURE(LENGTH,LOAD$ADDR,REC$TYPE);
  74    2                DECLARE (LENGTH,REC$TYPE) BYTE, LOAD$ADDR WORD;
  75    2                CALL SIO$OUT$CHAR(':');
  76    2                CHECK$SUM = 0;
  77    2                CALL SIO$OUT$BYTE(LENGTH);
  78    2                CALL SIO$OUT$WORD(LOAD$ADDR);
  79    2                CALL SIO$OUT$BYTE(REC$TYPE);
  80    2             END;

  81    1          SIO$8251$SETTLING$DELAY:
                     /* THIS DELAY ROUTINE ALLOWS THE USART SUFFICIENT TIME
                        TO SETTLE AFTER WRITE OPERATIONS. */
                     PROCEDURE;
  82    2                I = SHR(0FFH,0FFH);
  83    2             END;


                      /*********************************
                      *   UTILITY ROUTINES SECTION    *
```

```
                    ********************************/

  84   1        SIO$VALID$HEX:
                   /* THIS ROUTINE TESTS IF THE INPUT PARM IS A VALID ASCII HEX DIGIT
                      AND RETURNS TRUE AS THE VALUE OF THE PROCEDURE IF SO AND FALSE
                      IF NOT. */
                   PROCEDURE (H) BYTE;
  85   2                DECLARE H BYTE;
  86   2                DO I=0 TO LAST(ASCII);
  87   3                   IF H=ASCII(I) THEN RETURN TRUE;
  89   3                END;
  90   2                RETURN FALSE;
  91   2             END;

  92   1        SIO$HEX:
                  /* THIS ROUTINE CONVERTS THE INPUT PARM FROM ASCII TO ITS BINARY
                     EQUIVALENT AND RETURNS IT AS THE VALUE OF THE PROCEDURE. NO CHECK
                     IS MADE FOR INPUT VALIDITY. */
                  PROCEDURE(C) WORD;
  93   2                DECLARE C BYTE;
  94   2                IF C<='9' THEN RETURN DOUBLE(C-30H);
  96   2                ELSE RETURN DOUBLE(C-37H);
  97   2             END;

  98   1        SIO$VALID$REG$FIRST:
                   /* THIS ROUTINE CHECKS IF 'CHAR' IS A VALID FIRST LETTER OF A REGISTER
                      NAME AND RETURNS TRUE AS THE VALUE OF THE PROCEDURE IF SO. */
                   PROCEDURE BYTE;
  99   2                DO I=0 TO 26 BY 2;
 100   3                   IF CHAR=REG(I) THEN RETURN TRUE;
 102   3                END;
 103   2                RETURN FALSE;
 104   2             END;

 105   1        SIO$VALID$REG:
                   /* THIS ROUTINE CHECKS IF THE TWO INPUT PARMS TAKEN TOGETHER FORM
                      A VALID REGISTER NAME.  IT SEARCHES THE REGISTER TABLE AND IF A
                      MATCH IS FOUND, THE GLOBAL 'REG$INDEX' IS SET TO THE INDEX OF THE
                      VALID REGISTER AND THE PROCEDURE RETURNS TRUE.  IF NO MATCH THE
                      PROCEDURE RETURNS FALSE AND REG$INDEX IS UNDEFINED. */
                   PROCEDURE (C1,C2) BYTE;
 106   2                DECLARE (C1,C2) BYTE;
 107   2                DO REG$INDEX=0 TO 13;
 108   3                   IF C1=REG(REG$INDEX*2) AND C2=REG(REG$INDEX*2+1) THEN
 109   3                      RETURN TRUE;
 110   3                END;
 111   2                RETURN FALSE;
 112   2             END;

 113   1        SIO$CRLF:
                   /* THIS ROUTINE OUTPUTS A CR AND LF TO THE OUTPUT PORT. */
                   PROCEDURE;
 114   2                CALL SIO$OUT$CHAR(ASCR);
 115   2                CALL SIO$OUT$CHAR(ASLF);
 116   2             END;

 117   1        SIO$TEST$WORD$MODE:
```

```
                  /* THIS PROCEDURE TESTS FOR A 'W' FOLLOWING THE COMMAND AND IF SO
                     SETS THE FLAG 'WORD$MODE TO TRUE OR FALSE OTHERWISE. SCANS OFF
                     OPTIONAL BLANK FOLLOWING COMMAND. */
                  PROCEDURE;
118   2              WORD$MODE = FALSE;
119   2              CALL SIO$GET$CHAR;
120   2              IF CHAR='W' THEN
121   2                 DO;
122   3                 WORD$MODE = TRUE;
123   3                 CALL SIO$GET$CHAR;
124   3                 END;
125   2              IF CHAR=ASBL THEN
126   2                 CALL SIO$GET$CHAR;
127   2           END;

128   1        SIO$SCAN$BLANK:
                  /* THIS ROUTINE IS CALLED AFTER A COMMAND LETTER TO SCAN OFF THE
                     OPTIONAL BLANK. */
                  PROCEDURE;
129   2              CALL SIO$GET$CHAR;
130   2              IF CHAR=ASBL THEN
131   2                 CALL SIO$GET$CHAR;
132   2           END;


                  /*********************************************
                  *    ARGUMENT EXPRESSION EVALUATOR SECTION    *
                  *********************************************/

133   1        SIO$GET$WORD:
                  /* THIS ROUTINE READS CHARS FROM THE INPUT PORT AND EVALUATES
                     AN EXPRESSION CONSISTING OF '+-' AND OPERANDS OF HEX NUMBERS
                     AND REGISTER NAMES. */
                  PROCEDURE WORD;
134   2              DECLARE (SAVE,W) WORD, (OPER,T) BYTE;
135   2              OPER = '+';
136   2              W = 0;
137   2              DO WHILE TRUE;
138   3                 T = CHAR;
139   3                 SAVE = 0;
140   3                 IF SIO$VALID$REG$FIRST THEN
141   3                    DO;
142   4                    CALL SIO$GET$CHAR;
143   4                    IF SIO$VALID$REG(T,CHAR) THEN
144   4                       DO;
145   5                       SAVE = REG$SAV(REG$INDEX);
146   5                       CALL SIO$GET$CHAR;
147   5                       GOTO EVAL;
148   5                       END;
                           ELSE
149   4                       SAVE = SIO$HEX(T);
150   4                    END;
151   3                 IF NOT(SIO$VALID$HEX(T)) THEN  GOTO ERROR;
153   3                 DO WHILE SIO$VALID$HEX(CHAR);
154   4                    SAVE = SHL(SAVE,4) + SIO$HEX(CHAR);
155   4                    CALL SIO$GET$CHAR;
156   4                 END;
```

```
157   3       EVAL:    IF OPER='+' THEN
158   3                   W = W + SAVE;
                       ELSE
159   3                   W = W - SAVE;
160   3                 IF CHAR=ASCR OR CHAR=':' OR CHAR=',' THEN
161   3                    RETURN W;
162   3                 IF CHAR='+' OR CHAR='-' THEN
163   3                    OPER = CHAR;
                       ELSE
164   3                    GOTO ERROR;
165   3                 CALL SIO$GET$CHAR;
166   3              END;
167   2           END;

168   1       SIO$GET$ADDR:
                  /* THIS ROUTINE ACCEPTS A VALID ADDRESS EXPRESSION CONSISTING
                     OF AN OPTIONAL <SEG>: AND AN DISPLACEMENT. */
                  PROCEDURE(PTR,DEFAULT$BASE);
169   2              DECLARE PTR POINTER, DEFAULT$BASE WORD,
                        ARG BASED PTR STRUCTURE (OFF WORD, SEG WORD);
170   2              ARG.SEG = DEFAULT$BASE;
171   2              ARG.OFF = SIO$GET$WORD;
172   2              IF CHAR=':' THEN
173   2                 DO;
174   3                 CALL SIO$GET$CHAR;
175   3                 ARG.SEG = ARG.OFF;
176   3                 ARG.OFF = SIO$GET$WORD;
177   3                 IF CHAR=':' THEN  GOTO ERROR;
179   3                 END;
180   2           END;

181   1       SIO$UPDATE$IP:
                  /* THIS PROCEDURE IS CALLED BY SINGLE STEP AND GO TO OUTPUT THE CURRENT
                     IP AND INSTRUCTION BYTE AND OPEN THE IP FOR INPUT. */
                  PROCEDURE;
182   2             CALL SIO$OUT$BLANK;
183   2             CALL SIO$OUT$WORD(IP);
184   2             CSIP.SEG = CS;
185   2             CSIP.OFF = IP;
186   2             CALL SIO$OUT$CHAR('-');
187   2             CALL SIO$OUT$BLANK;
188   2             CALL SIO$OUT$BYTE(MEMORY$CSIP);
189   2             CALL SIO$OUT$BLANK;
190   2             CALL SIO$GET$CHAR;
191   2             IF CHAR<>',' AND CHAR<>ASCR THEN CALL SIO$GET$ADDR(@CSIP,CS);
193   2           END;


                  /*****************************
                  *   PAPER TAPE READ SECTION   *
                  *****************************/

194   1       SIO$READ$CHAR:
                  /* THIS PROCEDURE READS A BYTE FROM THE PAPER TAPE READER OF THE TTY
                     BY ACTIVATING DTR, SAMPLING DSR FOR THE START BIT, DEACTIVATING
                     DTR WHEN THE START BIT APPEARS, AND READING THE DATUM FROM THE
                     USART. TO PROVIDE A HOLD-OFF CAPABILITY FOR THE LOAD COMMAND
```

```
                     ONLY (NOT INTENDED FOR TTY PAPER TAPE) CONTROL-S WILL CAUSE THIS
                     ROUTINE TO WAIT FOR A CORRESPONDING CONTROL-Q TO BE READ BEFORE
                     CONTINUING. */
                  PROCEDURE BYTE;
195   2              DECLARE DELAY WORD;
196   2           LOOP:
                     OUTPUT(SIO$STAT$PORT) = SIO$8251$DTR$ON;          /* DTR ON */
197   2              DELAY = 0;
198   2              DO WHILE (INPUT(SIO$STAT$PORT) AND SIO$DSRDY)=0; /*WAIT STARTBIT*/
199   3                 DELAY = DELAY + 1;
200   3                 IF DELAY>=MAX$DELAY THEN
201   3                    DO;
202   4                    OUTPUT(SIO$STAT$PORT) = SIO$8251$CMND;     /* DTR OFF */
203   4                    GOTO ERROR;
204   4                    END;
205   3                 END;
206   2              OUTPUT(SIO$STAT$PORT) = SIO$8251$CMND;           /* DTR OFF */
207   2              DO WHILE (INPUT(SIO$STAT$PORT) AND SIO$RXRDY)=0;END;  /* WAIT LOOP */
209   2              CALL SIO$CHECK$CONTROL$CHAR;
210   2              IF CHAR = 11H THEN GOTO LOOP;  /* GET ANOTHER IF CTL-Q */
212   2              RETURN CHAR;
213   2           END;

214   1        SIO$READ$BYTE:
                  /* THIS ROUTINE READS A BYTE FROM THE PAPER TAPE READER. */
                  PROCEDURE BYTE;
215   2              DECLARE T BYTE;
216   2              T = LOW(SIO$HEX(SIO$READ$CHAR));
217   2              T = SHL(T,4) + LOW(SIO$HEX(SIO$READ$CHAR));
218   2              CHECK$SUM = CHECK$SUM + T;
219   2              RETURN T;
220   2           END;

221   1        SIO$READ$WORD:
                  /* THIS ROUTINE READS A WORD FROM THE PAPER TAPE READER AND RETURNS IT
                     AS THE VALUE OF THE PROCEDURE. */
                  PROCEDURE WORD;
222   2              DECLARE T BYTE;
223   2              T = SIO$READ$BYTE;
224   2              RETURN SHL(DOUBLE(T),8) + DOUBLE(SIO$READ$BYTE);
225   2           END;


                  /*********************************************
                  *   INTERRUPT AND RESTORE/EXECUTE SECTION    *
                  *********************************************/

226   1        SAVE$REGISTERS:
                  /* THIS ROUTINE IS USED TO SAVE THE STACKED USER'S REGISTERS IN THE
                     MONITOR'S SAVE AREA. */
                  PROCEDURE;
227   2              BP = MEMORY$USERSTACK;
228   2              USERSTACK.OFF = USERSTACK.OFF + 4;
229   2              DO I=0 TO 10;                        /* POP REGISTERS OFF OF STACK */
230   3                 REG$SAV(REG$ORD(I)) = MEMORY$USERSTACK;
231   3                 USERSTACK.OFF = USERSTACK.OFF + 2;
232   3              END;
```

```
233   2             SS = USERSTACK.SEG;
234   2             SP = USERSTACK.OFF;
235   2          END;

236   1       RESTORE$EXECUTE:
             /* THIS PROCEDURE RESTORES THE STATE OF THE USER MACHINE AND
                PASSES CONTROL BACK TO THE USER PROGRAM. IT CONTAINS A
                MACHINE LANGUAGE SUBROUTINE TO PERFORM THE POPPING OF THE
                USER REGISTERS AND TO EXECUTE AN 'IRET' TO TRANSFER CONTROL
                TO THE USER'S PROGRAM. */
             PROCEDURE;
237   2          DECLARE RESTORE$EXECUTE$CODE(*) BYTE DATA
                   (08BH,0ECH,           /* MOV BP,SP            */
                    08BH,046H,002H,      /* MOV AX,/BP/.PARM2    */
                    08BH,05EH,004H,      /* MOV BX,/BP/.PARM1    */
                    08EH,0D0H,           /* MOV SS,AX            */
                    08BH,0E3H,           /* MOV SP,BX            */
                    05DH,                /* POP BP               */
                    05FH,                /* POP DI               */
                    05EH,                /* POP SI               */
                    05BH,                /* POP BX               */
                    05AH,                /* POP DX               */
                    059H,                /* POP CX               */
                    058H,                /* POP AX               */
                    01FH,                /* POP DS               */
                    007H,                /* POP ES               */
                    0CFH),               /* IRET                 */
                  RESTORE$EXECUTE$CODE$PTR WORD DATA (.RESTORE$EXECUTE$CODE);

238   2          USERSTACK.SEG = SS;
239   2          USERSTACK.OFF = SP;
240   2          DO I=0 TO 10;           /* PUSH USER'S REGISTERS ONTO HIS STACK */
241   3             USERSTACK.OFF = USERSTACK.OFF - 2;
242   3             MEMORY$USERSTACK = REG$SAV(REG$ORD(10-I));
243   3          END;
244   2          USERSTACK.OFF = USERSTACK.OFF - 2;
245   2          MEMORY$USERSTACK = BP;
246   2          CALL RESTORE$EXECUTE$CODE$PTR(USERSTACK.OFF,USERSTACK.SEG);
247   2       END;


248   1       INTERRUPT1$ENTRY:
             /* THIS PROCEDURE IS CALLED WHEN THE CPU IS INTERRUPTED BY EXECUTING
                AN INSTRUCTION WITH THE TRAP BIT SET (SINGLE STEP). */
             PROCEDURE INTERRUPT 1;
249   2          USERSTACK.OFF = STACKPTR;         /* CHANGE TO MONITOR'S STACK */
250   2          USERSTACK.SEG = STACKBASE;
251   2          STACKPTR =  MONITOR$STACKPTR;
252   2          STACKBASE = MONITOR$STACKBASE;
253   2          CALL SAVE$REGISTERS;
254   2          FL = FL AND (NOT STEP$TRAP);     /* CLEAR STEP FLAG */
255   2          IF LAST$COMMAND<>SS$COMMAND THEN  /* CONTINUE IF NOT SS */
256   2             CALL RESTORE$EXECUTE;
257   2          CALL SIO$CRLF;
258   2          CALL SIO$UPDATE$IP;
259   2          IF CHAR=',' THEN
260   2             DO;
```

```
261   3                IP = CSIP.OFF;
262   3                CS = CSIP.SEG;
263   3                FL = FL OR STEP$TRAP;          /* SET STEP FLAG */
264   3                CALL RESTORE$EXECUTE;
265   3                END;
266   2             IF CHAR<>ASCR THEN  GOTO ERROR;
268   2             GOTO NEXT$COMMAND;
269   2          END;

270   1       INTERRUPT3$ENTRY:
             /* THIS PROCEDURE IS CALLED WHEN THE CPU EXECUTES A 'INT 3' INSTRUCTION.
                THE MONITOR INSERTS THIS (0CCH) FOR A BREAKPOINT.  ALSO AN EXTERNAL
                INTERRUPT OR A USER SOFTWARE INTERRUPT MAY CAUSE THIS PROCEDURE TO BE
                CALLED. */
             PROCEDURE INTERRUPT 3;
271   2          USERSTACK.OFF = STACKPTR;
272   2          USERSTACK.SEG = STACKBASE;
273   2          STACKPTR =  MONITOR$STACKPTR;
274   2          STACKBASE = MONITOR$STACKBASE;
275   2          CALL SAVE$REGISTERS;
276   2          CALL SIO$CRLF;
277   2          GOTO AFTER$INTERRUPT;
278   2       END;

279   1       INIT$INT$VECTOR:
             /* THIS ROUTINE INITIALIZES AN INTERRRUPT VECTOR AS FOLLOWS: THE OFFSET
                FROM THE ADDRESS OF 'INT$ROUTINE' CORRECTED BY THE APPROPRIATE
                NUMBER OF BYTES FOR THE INTERRUPT PLM PROLOGUE. THE SEGMENT FROM THE
                CURRENT CS REGISTER IS DETERMINED BY A MACHINE LANGUAGE CODED
                SUBROUTINE. */
             PROCEDURE(INT$VECTOR$PTR,INT$ROUTINE$OFFSET);
280   2          DECLARE INT$VECTOR$PTR POINTER, INT$ROUTINE$OFFSET WORD,
                   VECTOR BASED INT$VECTOR$PTR STRUCTURE (OFF WORD, SEG WORD),
                   CORRECTION LITERALLY '19H',  /* OFFSET FOR PROLOGUE */
                   INIT$INT$VECTOR$CODE(*) BYTE DATA
                   (055H,                              /* PUSH BP            */
                    08BH,0ECH,                         /* MOV BP,SP          */
                    08CH,0C8H,                         /* MOV AX,CS          */
                    0C4H,05EH,004H,                    /* LES BX,/BP/.PARM1  */
                    026H,089H,007H,                    /* MOV ES:W/BX/,AX    */
                    05DH,                              /* POP BP             */
                    0C2H,004H,000H),                   /* RET 4              */
                   INIT$INT$VECTOR$CODE$PTR WORD DATA (.INIT$INT$VECTOR$CODE);

281   2          CALL INIT$INT$VECTOR$CODE$PTR(@VECTOR.SEG);    /* SEGMENT PORTION */
282   2          VECTOR.OFF = INT$ROUTINE$OFFSET - CORRECTION;   /* OFFSET PORTION */
283   2       END;


          /* ************************************************************************
             ************************************************************************

                                 COMMAND MODULE
                                 ==============
             ABSTRACT
             ========
             THIS MODULE CONTAINS ALL THE COMMANDS IMPLEMENTED AS INDIVIDUAL PROCEDURES
```

```
                AND CALLED FROM THE OUTER BLOCK OF THE COMMAND DISPATCH LOOP.

                MODULE ORGANIZATION
                ===================
                THIS MODULE CONTAINS THE FOLLOWING SECTIONS:

                   1. COMMANDS SECTION
                       SIO$GO              GO
                       SIO$SINGLE$STEP     SINGLE STEP
                       SIO$EXAM$MEM        SUBSTITUTE MEMORY
                       SIO$EXAM$REG        EXAMINE REGISTER
                       SIO$MOVE            MOVE
                       SIO$DISPLAY         DISPLAY BYTES
                       SIO$INPUT           INPUT PORT
                       SIO$OUTPUT          OUTPUT PORT
                       SIO$WRITE           WRITE DATA RECORDS
                       SIO$READ            READ DATA RECORDS
                     2. COMMAND DISPATCH (OUTER BLOCK, MAIN PROGRAM LOOP)
                       NEXT$COMMAND        DISPATCH
                       ERROR               ERROR ROUTINE
             */

                /************************
                *   COMMAND SECTION    *
                ************************/


 284   1        SIO$GO:
                   /* IMPLEMENTS THE 'GO' COMMAND. THE USER MAY SPECIFY A NEW
                      IP:PC AND AN OPTIONAL BREAKPOINT. */
                   PROCEDURE;
 285   2            CALL SIO$UPDATE$IP;
 286   2            IF CHAR=',' THEN              /* BREAKPOINT */
 287   2               DO;
 288   3               CALL SIO$GET$CHAR;
 289   3               CALL SIO$GET$ADDR(@BRK1,CSIP.SEG);
 290   3               IF CHAR<>ASCR THEN  GOTO ERROR;
 292   3               BRK1$SAVE = MEMORY$BRK1;
 293   3               MEMORY$BRK1 = BREAK$INST;
 294   3               IF MEMORY$BRK1<>BREAK$INST THEN GOTO ERROR;
 296   3               BRK1$FLAG = TRUE;
 297   3               END;
                   ELSE                          /* NO BREAKPOINT */
 298   2               IF CHAR<>ASCR THEN  GOTO ERROR;
                   CALL SIO$CRLF;
 301   2            IP = CSIP.OFF;
 302   2            CS = CSIP.SEG;
 303   2            FL = FL AND (NOT STEP$TRAP);      /* CLEAR IF SET */
 304   2            CALL RESTORE$EXECUTE;
 305   2         END;

 306   1        SIO$SINGLE$STEP:
                   /* IMPLEMENTS THE SINGLE STEP COMMAND. DISPLAYS IP AND THE
                      CURRENT INSTRUCTION BTYE. OPENS CS:IP FOR INPUT. DEPRESSING
                      COMMA CAUSES THE MONITOR TO SINGLE STEP THE INSTRUCTION, AND
                      PERIOD TERMINATES THE COMMAND. */
                   PROCEDURE;
```

```
 307   2             CALL SIO$UPDATE$IP;
 308   2             IF CHAR<>',' THEN GOTO ERROR;
 310   2             IP = CSIP.OFF;
 311   2             CS = CSIP.SEG;
 312   2             FL = FL OR STEP$TRAP;
 313   2             CALL RESTORE$EXECUTE;
 314   2          END;

 315   1       SIO$EXAM$MEM:
                   /* IMPLEMENTS THE EXAMINE MEMORY COMMAND. */
                   PROCEDURE;
 316   2             DECLARE W WORD;
 317   2             CALL SIO$TEST$WORD$MODE;
 318   2             CALL SIO$GET$ADDR(@ARG1,CS);
 319   2             IF CHAR<>',' THEN  GOTO ERROR;
 321   2             DO WHILE TRUE;
 322   3                CALL SIO$OUT$BLANK;
 323   3                IF WORD$MODE THEN
 324   3                   CALL SIO$OUT$WORD(MEMORY$WORD$ARG1);
                      ELSE
 325   3                   CALL SIO$OUT$BYTE(MEMORY$ARG1);
 326   3                CALL SIO$OUT$CHAR('-');
 327   3                CALL SIO$OUT$BLANK;
 328   3                CALL SIO$GET$CHAR;
 329   3                IF CHAR=ASCR THEN RETURN;
 331   3                IF CHAR<>',' THEN
 332   3                   DO;
 333   4                   W = SIO$GET$WORD;
 334   4                   IF (CHAR <> ',') AND (CHAR <> ASCR) THEN GOTO ERROR;
 336   4                   IF WORD$MODE THEN
 337   4                      DO;
 338   5                      MEMORY$WORD$ARG1 = W;
 339   5                      IF MEMORY$WORD$ARG1<>W THEN GOTO ERROR;
 341   5                      END;
                         ELSE
 342   4                      DO;
 343   5                      MEMORY$ARG1 = LOW(W);
 344   5                      IF MEMORY$ARG1<>LOW(W) THEN GOTO ERROR;
 346   5                      END;
 347   4                   END;
 348   3                IF CHAR=ASCR THEN RETURN;
 350   3                IF WORD$MODE THEN
 351   3                   ARG1.OFF = ARG1.OFF + 2;
                      ELSE
 352   3                   ARG1.OFF = ARG1.OFF + 1;
 353   3                CALL SIO$CRLF;
 354   3                CALL SIO$OUT$WORD(ARG1.OFF);
 355   3             END;
 356   2          END;

 357   1       SIO$EXAM$REG:
                   /* IMPLEMENTS THE EXAMINE REGISTER COMMAND. SCANS FOR A VALID
                      REGISTER NAME AND DISPLAYS THE VALUE OF THAT REGISTER WHICH IS
                      OPTIONALLY OPENED FOR INPUT. COMMA INCREMENTS TO NEXT REGISTER
                      UNLESS IT IS 'FL' WHICH TERMINATES AS DOES CR. */
                   PROCEDURE;
 358   2             DECLARE (T,I) BYTE, SAVE WORD;
```

```
 359   2             CALL SIO$SCAN$BLANK;
 360   2             IF CHAR=ASCR THEN
 361   2                DO;
 362   3                CALL SIO$CRLF;
 363   3                DO I=0 TO 13;
 364   4                   CALL SIO$OUT$BLANK;
 365   4                   CALL SIO$OUT$CHAR(REG(I*2));
 366   4                   CALL SIO$OUT$CHAR(REG(I*2+1));
 367   4                   CALL SIO$OUT$CHAR('=');
 368   4                   CALL SIO$OUT$WORD(REG$SAV(I));
 369   4                   IF I=6 THEN CALL SIO$CRLF;
 371   4                END;
 372   3                RETURN;
 373   3                END;
 374   2             IF NOT(SIO$VALID$REG$FIRST) THEN  GOTO ERROR;
 376   2             T = CHAR;
 377   2             CALL SIO$GET$CHAR;
 378   2             IF NOT(SIO$VALID$REG(T,CHAR)) THEN  GOTO ERROR;
 380   2             I = REG$INDEX;
 381   2             DO WHILE TRUE;
 382   3                CALL SIO$OUT$CHAR('=');
 383   3                CALL SIO$OUT$WORD(REG$SAV(I));
 384   3                CALL SIO$OUT$CHAR('-');
 385   3                CALL SIO$OUT$BLANK;
 386   3                CALL SIO$GET$CHAR;
 387   3                IF CHAR<>',' AND CHAR<>ASCR THEN
 388   3                   DO;
 389   4                   SAVE = SIO$GET$WORD;
 390   4                   IF (CHAR <> ',' ) AND (CHAR <> ASCR) THEN GOTO ERROR;
 392   4                   REG$SAV(I) = SAVE;
 393   4                   END;
 394   3                IF CHAR=ASCR OR I=13 THEN RETURN;
 396   3                I = I + 1;
 397   3                CALL SIO$CRLF;
 398   3                CALL SIO$OUT$CHAR(REG(I*2));
 399   3                CALL SIO$OUT$CHAR(REG(I*2+1));
 400   3             END;
 401   2          END;

 402   1       SIO$MOVE:
                /* IMPLEMENTS THE MOVE COMMAND. ACCEPTS 3 ARGUMENTS AMD MOVES THE
                   BLOCK OF MEMORY SPECIFIED BY ARG1-ARG2 TO ARG3. ARG2<ARG1 OR THERE
                   IS A DIFFERENCE WHEN THE BYTE IS READ BACK, THEN ERROR. */
             PROCEDURE;
 403   2          CALL SIO$SCAN$BLANK;
 404   2          CALL SIO$GET$ADDR(@ARG1,CS);           /* FIRST ARGUMENT */
 405   2          IF CHAR<>',' THEN GOTO ERROR;
 407   2          CALL SIO$GET$CHAR;
 408   2          END$OFF = SIO$GET$WORD;                /* SECOND ARGUMENT */
 409   2          IF END$OFF<ARG1.OFF THEN  GOTO ERROR;
 411   2          IF CHAR<>',' THEN  GOTO ERROR;
 413   2          CALL SIO$GET$CHAR;
 414   2          CALL SIO$GET$ADDR(@ARG3,CS);     /* THIRD ARGUMENT */
 415   2          IF CHAR<>ASCR THEN GOTO ERROR;
 417   2          CALL SIO$CRLF;
 418   2       LOOP:
                   MEMORY$ARG3 = MEMORY$ARG1;
```

```
419   2             IF MEMORY$ARG3<>MEMORY$ARG1 THEN GOTO ERROR;
421   2             IF ARG1.OFF = END$OFF THEN RETURN;
423   2             ARG1.OFF = ARG1.OFF + 1;
424   2             ARG3.OFF = ARG3.OFF + 1;
425   2             GOTO LOOP;
426   2          END;

427   1       SIO$DISPLAY:
                  /* IMPLEMENTS THE DISPLAY BYTE COMMAND. IF CALLED WITH 1 PARM THEN
                       OUTPUTS A SINGLE BYTE. IF CALLED WITH 2 PARMS THEN OUTPUTS THE RANGE
                       BETWEEN THE TWO ADDRESSES. IF OFFSET<BEGIN THEN OUTPUTS ONLY A SINGLE
                       BYTE. */
                  PROCEDURE;
428   2             DECLARE T BYTE;
429   2             CALL SIO$TEST$WORD$MODE;
430   2             CALL SIO$GET$ADDR(@ARG1,CS);
431   2             IF CHAR=ASCR THEN
432   2                END$OFF = ARG1.OFF;
                    ELSE
433   2                DO;
434   3                IF CHAR<>',' THEN GOTO ERROR;
436   3                CALL SIO$GET$CHAR;
437   3                END$OFF = SIO$GET$WORD;
438   3                IF END$OFF < ARG1.OFF THEN GOTO ERROR;
440   3                IF CHAR<>ASCR THEN  GOTO ERROR;
442   3                END;
443   2       NEWLINE:
                       CALL SIO$CRLF;
444   2             CALL SIO$OUT$WORD(ARG1.OFF);
445   2       LOOP: CALL SIO$OUT$BLANK;
446   2             IF WORD$MODE THEN
447   2                DO;
448   3                CALL SIO$OUT$WORD(MEMORY$WORD$ARG1);
449   3                IF ARG1.OFF = END$OFF THEN RETURN;
451   3                ARG1.OFF = ARG1.OFF + 1;
452   3                END;
                    ELSE
453   2                CALL SIO$OUT$BYTE(MEMORY$ARG1);
454   2             IF ARG1.OFF>=END$OFF THEN RETURN;
456   2             ARG1.OFF = ARG1.OFF + 1;
457   2             T = ARG1.OFF AND 000FH;
458   2             IF T=0 OR (WORD$MODE AND T=1) THEN GOTO NEWLINE;
460   2             GOTO LOOP;
461   2             END;

462   1       SIO$INPUT:
                  /* THIS ROUTINE IMPLEMENTS THE 'INPUT' COMMAND. USER SPECIFIES
                     A PORT AND THE DATUM OF THE PORT IS DISPLAYED. */
                PROCEDURE;
463   2             DECLARE PORT WORD;
464   2             CALL SIO$TEST$WORD$MODE;
465   2             PORT = SIO$GET$WORD;
466   2       LOOP:
                    IF CHAR<>',' THEN GOTO ERROR;
468   2             CALL SIO$CRLF;
469   2             IF WORD$MODE THEN
470   2                CALL SIO$OUT$WORD(INWORD(PORT));
```

```
                  ELSE
 471   2                 CALL SIO$OUT$BYTE(INPUT(PORT));
 472   2              CALL SIO$GET$CHAR;
 473   2              IF CHAR=ASCR THEN RETURN;
 475   2              GOTO LOOP;
 476   2           END;

 477   1        SIO$OUTPUT:
              /* THIS ROUTINE IMPLEMENTS THE 'OUTPUT' COMMAND. THE USER SUPPLIED
                 DATUM IS OUTPUT TO THE SPECIFIED PORT. */
              PROCEDURE;
 478   2              DECLARE (DATUM,PORT) WORD;
 479   2              CALL SIO$TEST$WORD$MODE;
 480   2              PORT = SIO$GET$WORD;
 481   2              IF CHAR<>',' THEN GOTO ERROR;
 483   2              CALL SIO$GET$CHAR;
 484   2        LOOP:
                    DATUM = SIO$GET$WORD;
 485   2              IF CHAR=':' THEN GOTO ERROR;
 487   2              IF WORD$MODE THEN
 488   2                 OUTWORD(PORT) = DATUM;
                    ELSE
 489   2                 OUTPUT(PORT) = LOW(DATUM);
 490   2              IF CHAR=',' THEN
 491   2                 DO;
 492   3                 CALL SIO$CRLF;
 493   3                 CALL SIO$OUT$CHAR('-');
 494   3                 CALL SIO$OUT$BLANK;
 495   3                 CALL SIO$GET$CHAR;
 496   3                 IF CHAR <> ASCR THEN GOTO LOOP;
 498   3                 END;
 499   2           END;

 500   1        SIO$WRITE:
              /* IMPLEMENTS THE PAPER TAPE WRITE COMMAND. OUTPUTS LEADING NULLS,
                  EXTENDED ADDRESS RECORD (8086 ONLY), START ADDRESS RECORDS
                  (8086 ONLY), DATA RECORDS, EOF RECORD, AND TRAILING NULLS. */
              PROCEDURE;
 501   2              DECLARE (START$REC,MODE$8086) BYTE, (LEN,INDEX) WORD;
 502   2              CALL SIO$GET$CHAR;
 503   2              MODE$8086 = TRUE;
 504   2              IF CHAR='X' THEN                  /* TEST FOR 8080 MODE */
 505   2                 DO;
 506   3                 MODE$8086 = FALSE;
 507   3                 CALL SIO$GET$CHAR;
 508   3                 END;
 509   2              IF CHAR=ASBL THEN CALL SIO$GET$CHAR;
 511   2              CALL SIO$GET$ADDR(@ARG1,CS);
 512   2              IF CHAR<>',' THEN GOTO ERROR;
 514   2              CALL SIO$GET$CHAR;
 515   2              END$OFF = SIO$GET$WORD;
 516   2              IF END$OFF<ARG1.OFF THEN GOTO ERROR;
 518   2              IF CHAR<>ASCR THEN
 519   2                 DO;
 520   3                 START$REC = TRUE;
 521   3                 CALL SIO$GET$CHAR;
 522   3                 CALL SIO$GET$ADDR(@ARG3,CS);
```

```
523   3                 END;
                      ELSE
524   2                 DO;
525   3                 START$REC = FALSE;
526   3                 ARG3.OFF = 0;
527   3                 END;
528   2                 IF CHAR<>ASCR THEN GOTO ERROR;
530   2                 CALL SIO$CRLF;

531   2                 DO I=1 TO 60;              /* LEADING NULLS */
532   3                    CALL SIO$OUT$CHAR(0);
533   3                 END;
534   2                 CALL SIO$CRLF;

535   2                 IF MODE$8086 THEN
536   2                    DO;
537   3                    IF START$REC THEN
538   3                       DO;
539   4                       CALL SIO$OUT$HEADER(04,0,03); /* START ADDRESS RECORD */
540   4                       CALL SIO$OUT$WORD(ARG3.SEG);
541   4                       CALL SIO$OUT$WORD(ARG3.OFF);
542   4                       CALL SIO$OUT$BYTE(CHECK$SUM);
543   4                       CALL SIO$CRLF;
544   4                       ARG3.OFF = 0;
545   4                       END;
546   3                    CALL SIO$OUT$HEADER(02,0,02); /* EXTENDED ADDRESS RECORD */
547   3                    CALL SIO$OUT$WORD(ARG1.SEG);
548   3                    CALL SIO$OUT$BYTE(CHECK$SUM);
549   3                    CALL SIO$CRLF;
550   3                    END;

551   2                 LEN = STANDARD$LEN;              /* DATA RECORD */
552   2           LOOP: INDEX = END$OFF - ARG1.OFF;
553   2                 IF INDEX<STANDARD$LEN THEN LEN = INDEX + 1;
555   2                 CALL SIO$OUT$HEADER(LEN,ARG1.OFF,00);
556   2                 DO I=1 TO LEN;
557   3                    CALL SIO$OUT$BYTE(MEMORY$ARG1);
558   3                    ARG1.OFF = ARG1.OFF + 1;
559   3                 END;
560   2                 CALL SIO$OUT$BYTE(CHECK$SUM);
561   2                 CALL SIO$CRLF;
562   2                 IF END$OFF<>ARG1.OFF-1 THEN GOTO LOOP;

564   2                 CALL SIO$OUT$HEADER(00,ARG3.OFF,01);     /* EOF RECORD */
565   2                 CALL SIO$OUT$BYTE(CHECK$SUM);
566   2                 CALL SIO$CRLF;

567   2                 DO I=1 TO 60;            /* TRAILING NULLS */
568   3                    CALL SIO$OUT$CHAR(0);
569   3                 END;
570   2              END;

571   1           SIO$READ:
                    /* IMPLEMENTS THE READ PAPER TAPE COMMAND. */
                    PROCEDURE;
572   2                 DECLARE BIAS WORD, (REC$TYPE,LEN,I,T) BYTE, OFFSET WORD;
573   2                 BIAS = 0;                     /* DEFAULT BIAS */
```

```
574   2             ARG1.SEG = 0;              /* DEFAULT CS FOR 8080 FORMAT FILE */
575   2             CALL SIO$SCAN$BLANK;
576   2             IF CHAR<>ASCR THEN  BIAS = SIO$GET$WORD;
578   2             IF CHAR<>ASCR THEN GOTO ERROR;
580   2             CALL SIO$CRLF;
581   2       LOOP:
                    DO WHILE SIO$READ$CHAR<>':';END;
583   2             CHECK$SUM = 0;
584   2             LEN = SIO$READ$BYTE;
585   2             OFFSET = SIO$READ$WORD;
586   2             ARG1.OFF = OFFSET + BIAS;
587   2             REC$TYPE = SIO$READ$BYTE;
588   2             IF REC$TYPE=03 THEN        /* START ADDR TYPE */
589   2                DO;
590   3                CS = SIO$READ$WORD;
591   3                IP = SIO$READ$WORD;
592   3                END;
593   2             IF REC$TYPE=02 THEN        /* EXTENDED ADDR TYPE */
594   2                ARG1.SEG = SIO$READ$WORD;
595   2             IF REC$TYPE = 01 THEN     /* EOF */
596   2                IF OFFSET <> 0 THEN IP = OFFSET;
598   2             IF REC$TYPE=00 THEN        /* DATA TYPE */
599   2                DO I=1 TO LEN;
600   3                   T,MEMORY$ARG1 = SIO$READ$BYTE;
601   3                   IF MEMORY$ARG1<>T THEN GOTO ERROR;
603   3                   ARG1.OFF = ARG1.OFF + 1;
604   3                END;
605   2             T = SIO$READ$BYTE;         /* FETCH CHECKSUM */
606   2             IF CHECK$SUM<>0 THEN GOTO ERROR;
608   2             IF REC$TYPE<>01 AND LEN<>0 THEN GOTO LOOP;
610   2             CALL SIO$OUT$CHAR(0);  /* DELAY FOR LAST CR, LF SENT */
611   2             CALL SIO$OUT$CHAR(0);  /*     BY INTELLEC */
612   2          END;


                  /*****************************************
                  *   COMMAND DISPATCH MAIN PROGRAM LOOP   *
                  *****************************************/

613   1             DISABLE;
614   1             OUTPUT(KB$STAT$PORT) = KB$INIT$MODE;
615   1             OUTPUT(KB$STAT$PORT) = KB$INIT$SCAN;
616   1             OUTPUT(KB$STAT$PORT) = 90H; /* OUTPUT '8086' SIGNON BACKWARDS */
617   1             DO I=0 TO 7;
618   2                OUTPUT(KB$DATA$PORT) = KB$SIGNON(I);
619   2             END;

                  /* THIS STRANGE SEQUENCE OF CODE IS USED TO GUARANTEE CORRECT USART
                     INITIALIZATION AFTER EITHER: 1) HARDWARE RESET (EXPECTING MODE)
                     2) SOFTWARE RESTART (EXPECTING COMMAND). */

620   1             OUTPUT(SIO$STAT$PORT) = SIO$8251$RESET;
621   1             CALL SIO$8251$SETTLING$DELAY;
622   1             OUTPUT(SIO$STAT$PORT) = 25H;
623   1             CALL SIO$8251$SETTLING$DELAY;
624   1             OUTPUT(SIO$STAT$PORT) = SIO$8251$RESET;
625   1             CALL SIO$8251$SETTLING$DELAY;
```

```
626   1             OUTPUT(SIO$STAT$PORT) = SIO$8251$MODE;
627   1             CALL SIO$8251$SETTLING$DELAY;
628   1             OUTPUT(SIO$STAT$PORT) = SIO$8251$CMND;
629   1             CALL SIO$8251$SETTLING$DELAY;
630   1             CALL SIO$OUT$STRING(@SIO$SIGNON);

                    /* INITIALIZE USER'S REGISTERS */
631   1             CS,SS,DS,ES,FL,IP = 0;
632   1             SP = USER$INIT$SP;

                    /* INITIALIZE INTERRUPT VECTORS */
633   1             CALL INIT$INT$VECTOR(@INT$VECTOR(1),.INTERRUPT1$ENTRY);
634   1             CALL INIT$INT$VECTOR(@INT$VECTOR(2),.INTERRUPT3$ENTRY);
635   1             CALL INIT$INT$VECTOR(@INT$VECTOR(3),.INTERRUPT3$ENTRY);

636   1             BRK1$FLAG = FALSE;
637   1             MONITOR$STACKPTR = STACKPTR;
638   1             MONITOR$STACKBASE = STACKBASE;

639   1       NEXT$COMMAND:
                    /* THIS IS THE PERPETUAL COMMAND LOOP WHICH DISPATCHES TO EACH
                       COMMAND WHICH IS A SEPARATED PROCEDURE. */

                    CALL SIO$CRLF;
640   1             CALL SIO$OUT$CHAR(0);
641   1             CALL SIO$OUT$CHAR('.');
642   1             CALL SIO$GET$CHAR;
643   1             DO I=0 TO LAST(SIO$CMND);
644   2                IF CHAR=SIO$CMND(I) THEN GOTO DISPATCH;
646   2             END;
647   1             GOTO ERROR;
648   1       DISPATCH:
                    LAST$COMMAND = I;
649   1             DO CASE I;
650   2                CALL SIO$EXAM$MEM;
651   2                CALL SIO$EXAM$REG;
652   2                CALL SIO$GO;
653   2                CALL SIO$SINGLE$STEP;
654   2                CALL SIO$MOVE;
655   2                CALL SIO$DISPLAY;
656   2                CALL SIO$INPUT;
657   2                CALL SIO$OUTPUT;
658   2                CALL SIO$READ;
659   2                CALL SIO$WRITE;
660   2             END;
661   1             GOTO NEXT$COMMAND;

662   1       ERROR:
                    /* THIS ROUTINE HANDLES ALL ERRORS DETECTED BY THE MONITOR AND
                       WILL OUTPUT THE ERROR PROMPT TO THE OUTPUT PORT.*/

                    CALL SIO$OUT$CHAR('#');
663   1             GOTO NEXT$COMMAND;

664   1       AFTER$INTERRUPT:
                    /* THIS ROUTINE IS CALLED AFTER AN INTERRUPT TO DISPLAY THE CS:IP
                       AND RESTORE THE BREAKPOINTED INSTRUCTION. */
```

```
                    IF BRK1$FLAG THEN
 665   1                DO;
 666   2                MEMORY$BRK1 = BRK1$SAVE;
 667   2                BRK1$FLAG = FALSE;
 668   2                IF ((IP-1) AND 000FH)=(BRK1.OFF AND 000FH) AND
                             (SHR(IP-1,4)+CS)=(SHR(BRK1.OFF,4)+BRK1.SEG) THEN
 669   2                   DO;
 670   3                   IP = IP - 1;
 671   3                   CALL SIO$OUT$STRING(@SIO$BREAK$MSG);
 672   3                   END;
 673   2                END;
 674   1              CALL SIO$OUT$CHAR('@');
 675   1              CALL SIO$OUT$WORD(CS);
 676   1              CALL SIO$OUT$CHAR(':');
 677   1              CALL SIO$OUT$WORD(IP);
 678   1              GOTO NEXT$COMMAND;

 679   1           END MONITOR;       /* END OF MODULE */
            EOF
```

MODULE INFORMATION:

```
     CODE AREA SIZE     = 0FDBH   4059D
     CONSTANT AREA SIZE = 0000H      0D
     VARIABLE AREA SIZE = 0078H    120D
     MAXIMUM STACK SIZE = 0042H     66D
     1211 LINES READ
     0 PROGRAM ERROR(S)
```

END OF PL/M-86 COMPILATION

SDK-86 SERIAL MONITOR EXPANDED SOURCE LISTING


```
ISIS-II PL/M-86 V1.1 COMPILATION OF MODULE MONITOR
NO OBJECT MODULE REQUESTED
COMPILER INVOKED BY:  PLM86 :F1:SERIAL.PLM LARGE OPTIMIZE(2) PAGEWIDTH(95) &
                      CODE XREF NOOBJECT PRINT(:F1:SERIAL.PRT)


           $TITLE('SDK86 SERIAL MONITOR')
           $NOINTVECTOR

            /* ***********************************************************************
               ***********************************************************************

                    SDK-86 MONITOR, V1.2
                    06 AUGUST 1979

                    (C) 1978 INTEL CORPORATION. ALL RIGHTS RESERVED. NO PART
               OF THIS PROGRAM OR PUBLICATION MAY BE REPRODUCED, TRANSMITTED,
               TRANSCRIBED, STORED IN A RETRIEVAL SYSTEM, OR TRANSLATED INTO ANY
               LANGUAGE, IN ANY FORM OR BY ANY MEANS, ELECTRONIC, MECHANICAL,
               MAGNETIC, OPTICAL, CHEMICAL, MANUAL OR OTHERWISE, WITHOUT THE PRIOR
               WRITTEN PERMISSION OF INTEL CORPORATION, 3065 BOWERS AVENUE, SANTA
               CLARA, CALIFORNIA, 95051.

               ***********************************************************************
               ***********************************************************************

               ABSTRACT
               ========
               THIS PROGRAM IS THE ROM BASED SERIAL MONITOR FOR THE SDK-86.  IT
               PROVIDES THE USER WITH A MODERATE LEVEL OF CAPABLILITY TO
               EXAMINE/MODIFY MEMORY AND REGISTERS, CONTROL PROGRAM EXECUTION,
               AND LOAD/SAVE PROGRAMS.

               ENVIRONMENT
               ===========
               THE SDK SERIAL MONITOR COMMUNICATES WITH THE USER VIA AN INTERACTIVE
               TERMINAL (TTY,CRT) ATTACHED TO THE SERIAL PORT.

               PROGRAM ORGANIZATION
               ====================
               THE PROGRAM IS DIVIDED INTO 1 DATA AND 2 CODE MODULES:
                  1. DATA DECLARATION MODULE.    GLOBAL DATA DECLARATIONS.
                  2. COMMON ROUTINES.            LOWER LEVEL PROCEDURES
                  3. COMMAND MODULE.             INDIVIDUAL COMMANDS AND OUTER BLOCK

               CALLING GRAPH
               =============
                  >>COMMAND DISPATCH MODULE (OUTER BLOCK)
                       INDIVIDUAL COMMAND PROCEDURES
                          COMMON ROUTINES

               GLOBAL DATA STRUCTURES
               ======================
               THE MONITOR MAINTAINS THE USER'S MACHINE STATE (REGISTERS) IN A
               WORD ARRAY. THE REGISTERS ARE SAVED FROM THE USER'S STACK
               AS PUSHED BY PLM86 INTERRUPT PROCEDURE.
```

```
                  POINTERS TO THE SDK-86 2**20 ADDRESS SPACE ARE IMPLEMENTED WITH
                  POINTER STRUCTURES ALLOCATED AS 2 WORD STRUCTURES.

               */

1                 MONITOR:DO;                       /* BEGINNING OF MODULE */

               /* ***************************************************************
                  ***************************************************************

                                 GLOBAL DATA DECLARATIONS MODULE
                                 ===============================
                  ABSTRACT
                  ========
                  THIS MODULE CONTAINS ALL THE GLOBAL DATA DECLARATIONS AND
                  LITERALS (EQUATES).

                  MODULE ORGAINIZATION
                  ====================
                  THE MODULE IS DIVIDED INTO 5 SECTIONS:
                     1. UTILITY SECTION              GLOBAL FLAGS,VARIBLES,EQUATES
                     2. I/O SECTION                  I/O PORTS,MASKS,AND SPECIAL CHARS
                     3. MEMORY ARGUMENTS SECTIONS    STRUCTURES FOR POINTERS
                     4. REGISTER SECTION             USER REGISTER SAVE AREA
                     5. BOOT AND 8089 SECTION        BOOTSTRAP AND 8089 DESCRIPTOR
               */

                  /***********************
                  *   UTILITY SECTION   *
                  ***********************/

2    1          DECLARE
                       INT$VECTOR(5)      POINTER;            /* INTERRUPT VECTORS */
3    1          DECLARE
                       MONITOR$STACKPTR   WORD,
                       MONITOR$STACKBASE  WORD;
4    1          DECLARE
                       COPYRIGHT(*) BYTE DATA ('(C) 1978 INTEL CORP');
5    1          DECLARE
                       BRK1$FLAG          BYTE,               /* TRUE IF BREAK SET */
                       BRK1$SAVE          BYTE,               /* INST BREAK SAVE */
                       CHAR               BYTE,               /* ONE CHAR LOOK AHEAD */
                       CHECK$SUM          BYTE,               /* PAPER TAPE CHECKSUM */
                       I                  BYTE,               /* INDEX */
                       END$OFF            WORD,                /* END OFFSET ADDRESS */
                       WORD$MODE          BYTE,               /* WORD MODE FLAG */
                       LAST$COMMAND       BYTE;               /* LAST COMMAND SAVE */
6    1          DECLARE
                       TRUE               LITERALLY '0FFH',
                       FALSE              LITERALLY '000H',
                       BREAK$INST         LITERALLY '0CCH',   /* BREAKPOINT INST */
                       STEP$TRAP          LITERALLY '0100H',  /* SS TRAP FLAG MASK */
                       USER$INIT$SP       LITERALLY '100H',   /* USER STACK INITIAL */
                       GO$COMMAND         LITERALLY '2',      /* GO COMMAND CODE */
                       SS$COMMAND         LITERALLY '3',      /* SINGLE STEP CODE */
                       STANDARD$LEN       LITERALLY '16',     /* PAPER TAPE DATA REC LEN */
                       MAX$DELAY          LITERALLY '10000';  /* DELAY FOR PAPER TAPE */
```

```
 7   1       DECLARE
                 KB$SIGNON(*)     BYTE DATA  (5BH,86H,00H,00H,7DH,7FH,00H,00H),
                                        /* " 2  1.          6   8         " */
                 SIO$BREAK$MSG(*) BYTE DATA  ('BR ',0),
                 SIO$SIGNON(*)    BYTE DATA  (0DH,0AH,'SDK-86 MONITOR, V1.2',0),
                 ASCII(*)         BYTE DATA  ('0123456789ABCDEF'),
                 SIO$CMND(*)      BYTE DATA  ('SXGNMDIORW');

               /*******************************
               *   I/O DECLARATIONS SECTION   *
               *******************************/

 8   1       DECLARE                                        /* 8251 USART */
                 SIO$STAT$PORT   LITERALLY '0FFF2H',      /* STATUS INPUT PORT */
                 SIO$DATA$PORT   LITERALLY '0FFF0H',      /* DATA INPUT PORT */
                 SIO$8251$RESET  LITERALLY   '065H',      /* RESET COMMAND */
                 SIO$8251$MODE   LITERALLY   '0CFH',      /* ASYNC MODE */
                 SIO$8251$CMND   LITERALLY   '025H',      /* DTR OFF */
                 SIO$8251$DTR$ON LITERALLY   '027H',      /* SAME AS CMND BUT DTR ON */
                 SIO$DSRDY       LITERALLY    '80H',      /* DATA SET READY */
                 SIO$RXRDY       LITERALLY    '02H',      /* RECEIVE DATA READY */
                 SIO$TXRDY       LITERALLY    '01H';      /* XMIT BUFFER EMPTY */
 9   1       DECLARE
                 KB$STAT$PORT    LITERALLY '0FFEAH',      /* STATUS/COMMAND PORT */
                 KB$DATA$PORT    LITERALLY '0FFE8H',      /* DATA PORT      */
                 KB$INIT$MODE    LITERALLY '00H',         /* 8 8-BIT, LEFT ENTRY,
                                                             ENCODE, 2 KEY LOCKOUT */
                 KB$INIT$SCAN    LITERALLY '39H';         /* 10MS SCAN RATE       */
10   1       DECLARE
                 ASCR            LITERALLY  '0DH',        /* CARRIAGE RETURN */
                 ASLF            LITERALLY  '0AH',        /* LINE FEED       */
                 ASBL            LITERALLY  '20H';        /* BLANK OR SPACE  */

               /*******************************
               *   MEMORY ARGUMENT SECTION   *
               *******************************/

11   1       DECLARE
                 MEMORY$ARG1$PTR  POINTER,                /* ARGUMENT 1 */
                 ARG1 STRUCTURE (OFF WORD, SEG WORD)
                    AT (@MEMORY$ARG1$PTR),
                 MEMORY$ARG1 BASED MEMORY$ARG1$PTR BYTE,
                 MEMORY$WORD$ARG1 BASED MEMORY$ARG1$PTR WORD,

                 MEMORY$ARG3$PTR POINTER,                 /* ARGUMENT 3 */
                 ARG3 STRUCTURE (OFF WORD, SEG WORD)
                    AT (@MEMORY$ARG3$PTR),
                 MEMORY$ARG3 BASED MEMORY$ARG3$PTR BYTE,

                 MEMORY$BRK1$PTR POINTER,                 /* BREAKPOINT 1 */
                 BRK1 STRUCTURE (OFF WORD, SEG WORD)
                    AT (@MEMORY$BRK1$PTR),
                 MEMORY$BRK1 BASED MEMORY$BRK1$PTR BYTE,

                 MEMORY$CSIP$PTR POINTER,                 /* CS & IP WORD */
                 CSIP STRUCTURE (OFF WORD, SEG WORD)
                    AT (@MEMORY$CSIP$PTR),
```

```
                    MEMORY$CSIP BASED MEMORY$CSIP$PTR BYTE,

                    MEMORY$USERSTACK$PTR POINTER,
                    USERSTACK STRUCTURE (OFF WORD, SEG WORD)
                       AT (@MEMORY$USERSTACK$PTR),
                    MEMORY$USERSTACK BASED MEMORY$USERSTACK$PTR WORD;

                 /************************
                 *   REGISTER SECTION   *
                 ************************/

 12   1        DECLARE
                    REG(*)          BYTE DATA              /* REGISTER NAMES */
                       ('AXBXCXDXSPBPSIDICSDSSSESIPFL'),
                    REG$INDEX       WORD,
                    REG$SAV(14)     WORD,                /* USER'S SAVED REGS */
                    REG$ORD(*)      BYTE DATA
                       (7,6,1,3,2,0,9,11,12,8,13),
                    SP              LITERALLY 'REG$SAV( 4)',
                    BP              LITERALLY 'REG$SAV( 5)',
                    CS              LITERALLY 'REG$SAV( 8)',
                    DS              LITERALLY 'REG$SAV( 9)',
                    SS              LITERALLY 'REG$SAV(10)',
                    ES              LITERALLY 'REG$SAV(11)',
                    IP              LITERALLY 'REG$SAV(12)',
                    FL              LITERALLY 'REG$SAV(13)';


                 /*************************************
                 *   BOOTSTRAP JUMP AND 8089 VECTOR   *
                 *************************************/

                 /* THIS BOOT CONSISTS OF A LONG JUMP TO THE BEGINNING OF THE MONITOR
                    AT FF00+XXXX WHERE XXXX IS THE STARTING ADDRESS OFFSET (IP) AND
                    MUST BE DETERMINED AFTER EACH COMPILE. */
 13   1        DECLARE
                    START$ADDR LITERALLY '00A8H',                /* STARTING ADDRESS */
                    BOOT1(*)   BYTE AT (0FFFF0H) DATA (0EAH),    /* LONG JUMP OPCODE */
                    BOOT2(*)   WORD AT (0FFFF1H) DATA (START$ADDR),
                    BOOT3(*)   WORD AT (0FFFF3H) DATA (0FF00H); /* SEGMENT ADDRESS! */

                 /* THIS TWO-WORD DATA IS A JUMP TO THE STARTING ADDRESS AND IS LOCATED
                    AT THE FIRST LOCATION OF ROM (NO OTHER DATA OR CONSTANT DECLARATIONS
                    MAY PRECEDE IT). THE JUMP IS ACTUALLY TO (START-ADDR)-4 SINCE THE
                    INSTRUCTION IS A RELATIVE JUMP OF LENGTH 3. */
 14   1        DECLARE
                    BOOT4(*)   WORD DATA (0E990H,START$ADDR-4);  /* NOP,JMP START-ADDR */

                 /* THIS BLOCK OF ROM AT FFFF6-FFFFA IS INITIALIZED FOR THE 8089
                    DEVICE AND POINTS TO A BLOCK OF RAM AT 0100H. */
 15   1        DECLARE
                    BLOCK$8089      WORD    AT (0FFFF6H) DATA (00001H),
                    BLOCK$8089$PTR  POINTER AT (0FFFF8H) DATA (00100H);

                 /*********************************
                 *   FILL DECLARATIONS SECTION   *
                 *********************************/
```

```
            /* THESE DECLARATIONS DEFINE FILL CHARACTERS FOR ALL UNUSED
               LOCATIONS IN THE ROM ADDRESS SPACE. ANY MODIFICATION TO THE CODE
               THAT CHANGES THE USED LOCATIONS IN THE ROM REQUIRES
               MODIFICATION OF THIS SECTION. */

 16  1      DECLARE

              F        LITERALLY 'OFFH',
              FILL1(*) BYTE AT (OFFFDBH) DATA (F,F,F,F,F),
              FILL2(*) BYTE AT (OFFFE0H) DATA (F,F,F,F,F,F,F,F,F,F,F,F,F,F,F,F),
              FILL3(*) BYTE AT (OFFFF5H) DATA (F),
              FILL4(*) BYTE AT (OFFFFCH) DATA (F,F,F,F);


            /* ************************************************************************
               ************************************************************************

                              COMMON PROCEDURES
                              =================
               ABSTRACT
               ========
               THIS MODULE CONTAINS THOSE LOWER LEVEL PROCEDURES CALLED BY HIGHER
               LEVEL ROUTINES.

               MODULE ORGANIZATION
               ===================
               THIS MODULE IS DIVIDED INTO 4 SECTIONS AS FOLLOWS:
                  1. BASIC I/O SECTION
                      SIO$CHAR$RDY          INPUT CHARACTER READY
                      SIO$CHECK$CONTROL$CHAR   CHECK FOR CONTROL CHARACTER
                      SIO$OUT$CHAR          OUTPUT CHARACTER
                      SIO$GET$CHAR          INPUT A CHARACTER
                      SIO$OUT$BYTE          OUTPUT A BYTE IN HEX
                      SIO$OUT$WORD          OUTPUT A WORD IN HEX
                      SIO$OUT$BLANK         OUTPUT A SINGLE BLANK
                      SIO$OUT$STRING        OUTPUT A STRING
                      SIO$OUT$HEADER        OUTPUT A PAPER TAPE HEADER
                  2. UTILITY ROUTINES SECTION
                      SIO$VALID$HEX         TEST FOR VALID HEX CHAR
                      SIO$HEX               CONVERT TO HEX FROM ASCII
                      SIO$VALID$REG$FIRST   TEST FOR VALID REGISTER FIRST CHAR
                      SIO$VALID$REG         TEST FOR VALID REGISTER NAME
                      SIO$CRLF              OUTPUTS A CR AND LF
                      SIO$TEST$WORD$MODE    TEST FOR A 'W' IN COMMAND
                      SIO$SCAN$BLANK        SCANS FOR OPTIONAL BLANK
                  3. ARGUMENT EXPRESSION EVALUATION SECTION
                      SIO$GET$WORD          GET AN WORD EXPRESSION
                      SIO$GET$ADDR          GET AN ADDRESS EXPRESSION
                      SIO$UPDATE$IP         OPTIONAL UPDATE OF CS:IP
                  4. PAPER TAPE READ SECTION
                      SIO$READ$CHAR         READ CHAR FROM TTY READER
                      SIO$READ$BYTE         READ A BYTE
                      SIO$READ$WORD         READ A WORD
                  5. INTERRUPT AND RESTORE/EXECUTE ROUTINES
                      SAVE$REGISTERS        SAVES USER REGISTERS
                      RESTORE$EXECUTE       RESTORE MACHINE STATE AND EXEC
```

```
                        INTERRUPT1$ENTRY      INTERRUPT ROUTINE FOR SINGLE STEP
                        INTERRUPT3$ENTRY      INTERRUPT ROUTINE FOR GO
                        INIT$INT$VECTOR       INITIALIZES INTERRRUPT VECTORS
                *)

                   /************************
                   *   BASIC I/O SECTION   *
                   ************************/

 17   1        SIO$CHAR$RDY:
                                                ; STATEMENT # 17
                   /* TESTS FOR INPUT CHARACTER PENDING BY READING THE STATUS PORT
                      AND MASKING WITH SIO$RXRDY(READ DATA READY).  RETURNS TRUE IF
                      NOT EMPTY(CHAR PENDING) AND FALSE IF NO CHAR PENDING */
                       SIOCHARRDY       PROC NEAR
          02A0  55                      PUSH    BP
          02A1  8BEC                    MOV     BP,SP
                 PROCEDURE BYTE;
 18   2            IF (INPUT(SIO$STAT$PORT) AND SIO$RXRDY)=0 THEN RETURN FALSE;
                                                ; STATEMENT # 18
          02A3  BAF2FF                  MOV     DX,0FFF2H
          02A6  EC                      IN      DX
          02A7  A802                    TEST    AL,2H
          02A9  7504                    JNZ     @1
                                                ; STATEMENT # 19
          02AB  B000                    MOV     AL,0H
          02AD  5D                      POP     BP
          02AE  C3                      RET
                       @1:
 20   2            RETURN TRUE;
                                                ; STATEMENT # 20
          02AF  B0FF                    MOV     AL,0FFH
          02B1  5D                      POP     BP
          02B2  C3                      RET
 21   2             END;
                                                ; STATEMENT # 21
                       SIOCHARRDY       ENDP

 22   1        SIO$CHECK$CONTROL$CHAR:
                                                ; STATEMENT # 22
                   /* THIS ROUTINE CHECKS IF A CONTROL CHARACTER HAS BEEN INPUT TO
                      THE SERIAL PORT.  AFTER A CONTROL-S IT WAITS FOR A CONTROL-Q BEFORE
                      RETURNING TO THE CALLER.  A CONTROL-C CAUSES A JUMP TO THE ERROR
                      ROUTINE. */
                       SIOCHECKCONTROLCHAR      PROC NEAR
          02B3  55                      PUSH    BP
          02B4  8BEC                    MOV     BP,SP
                 PROCEDURE;
 23   2            CHAR = INPUT(SIO$DATA$PORT) AND 07FH;
                                                ; STATEMENT # 23
          02B6  BAF0FF                  MOV     DX,0FFF0H
          02B9  EC                      IN      DX
          02BA  247F                    AND     AL,7FH
          02BC  A26600                  MOV     CHAR,AL
 24   2            IF CHAR=13H THEN                /* CONTROL-S ? */
                                                ; STATEMENT # 24
          02BF  3C13                    CMP     AL,13H
```

```
                02C1  7520            JNZ     @2
25   2                DO WHILE CHAR<>11H;          /* CONTROL-Q   */
                                                  ; STATEMENT # 25
                      @121:
                02C3  803E660011      CMP     CHAR,11H
                02C8  7423            JZ      @5
26   3                   IF SIO$CHAR$RDY THEN
                                                  ; STATEMENT # 26
                02CA  E8D3FF          CALL    SIOCHARRDY
                02CD  D0D8            RCR     AL,1
                02CF  73F2            JNB     @121
27   3                      DO;
28   4                      CHAR = INPUT(SIO$DATA$PORT) AND 07FH;
                                                  ; STATEMENT # 28
                02D1  BAF0FF          MOV     DX,0FFF0H
                02D4  EC              IN      DX
                02D5  247F            AND     AL,7FH
                02D7  A26600          MOV     CHAR,AL
29   4                      IF CHAR=03H THEN GOTO ERROR;
                                                  ; STATEMENT # 29
                02DA  3C03            CMP     AL,3H
                02DC  7503            JNZ     $+5H
                02DE  E925FF          JMP     ERROR
31   4                      END;
32   3                END;
                                                  ; STATEMENT # 32
                02E1  EBE0            JMP     @121
                      @2:
33   2              ELSE IF CHAR = 03H THEN GOTO ERROR;
                                                  ; STATEMENT # 33
                02E3  803E660003      CMP     CHAR,3H
                02E8  7503            JNZ     $+5H
                02EA  E919FF          JMP     ERROR
                                                  ; STATEMENT # 34
                      @5:
                 END SIO$CHECK$CONTROL$CHAR;
                                                  ; STATEMENT # 35
                02ED  5D              POP     BP
                02EE  C3              RET
                      SIOCHECKCONTROLCHAR      ENDP

36   1       SIO$OUT$CHAR:
                                                  ; STATEMENT # 36
                 /* THIS ROUTINE OUTPUTS THE INPUT PARAMETER TO THE USART OUTPUT
                    PORT WHEN USART IS READY FOR OUTPUT (XMIT BUFFER EMPTY). */
                      SIOOUTCHAR      PROC NEAR
                02EF  55              PUSH    BP
                02F0  8BEC            MOV     BP,SP
                 PROCEDURE(C);
37   2              DECLARE C BYTE;
38   2              IF SIO$CHAR$RDY THEN CALL SIO$CHECK$CONTROL$CHAR;
                                                  ; STATEMENT # 38
                02F2  E8ABFF          CALL    SIOCHARRDY
                02F5  D0D8            RCR     AL,1
                02F7  7303            JNB     @123
                                                  ; STATEMENT # 39
                02F9  E8B7FF          CALL    SIOCHECKCONTROLCHAR
```

```
 40   2             DO WHILE (INPUT(SIO$STAT$PORT) AND SIO$TXRDY)=0;END;
                                                    ; STATEMENT # 40
                        @123:
          02FC  BAF2FF          MOV     DX,0FFF2H
          02FF  EC              IN      DX
          0300  A801            TEST    AL,1H
          0302  74F8            JZ      @123
 42   2             OUTPUT(SIO$DATA$PORT) = C;
                                                    ; STATEMENT # 42
          0304  BA0FF           MOV     DX,0FFF0H
          0307  8A4604          MOV     AL,[BP].C
          030A  EE              OUT     DX
 43   2           END;
                                                    ; STATEMENT # 43
          030B  5D              POP     BP
          030C  C20200          RET     2H
                        SIOOUTCHAR      ENDP

 44   1       SIO$GET$CHAR:
                                                    ; STATEMENT # 44
                  /* THIS ROUTINE INPUTS A CHARACTER FROM THE INPUT PORT AND RETURNS
                     WITH IT IN THE GLOBAL 'CHAR'. THE CHARACTER IS ECHOED TO THE
                     OUTPUT PORT IF PRINTABLE. */
                        SIOGETCHAR      PROC NEAR
          030F  55              PUSH    BP
          0310  8BEC            MOV     BP,SP
           PROCEDURE;
 45   2             DO WHILE (INPUT(SIO$STAT$PORT) AND SIO$RXRDY)=0;END;
                                                    ; STATEMENT # 45
                        @125:
          0312  BAF2FF          MOV     DX,0FFF2H
          0315  EC              IN      DX
          0316  A802            TEST    AL,2H
          0318  74F8            JZ      @125
 47   2             CHAR = INPUT(SIO$DATA$PORT) AND 07FH;
                                                    ; STATEMENT # 47
          031A  BA0FF           MOV     DX,0FFF0H
          031D  EC              IN      DX
          031E  247F            AND     AL,7FH
          0320  A26600          MOV     CHAR,AL
 48   2             IF CHAR>=ASBL THEN CALL SIO$OUT$CHAR(CHAR);
                                                    ; STATEMENT # 48
          0323  3C20            CMP     AL,20H
          0325  7207            JB      @8
                                                    ; STATEMENT # 49
          0327  FF366600        PUSH    CHAR    ; 1
          032B  E8C1FF          CALL    SIOOUTCHAR
                        @8:
 50   2           END;
                                                    ; STATEMENT # 50
          032E  5D              POP     BP
          032F  C3              RET
                        SIOGETCHAR      ENDP

 51   1       SIO$OUT$BYTE:
                                                    ; STATEMENT # 51
                  /* THIS ROUTINE OUTPUTS THE SINGLE INPUT PARAMETER TO THE USART
```

```
                    IN ASCII HEXADECIMAL FORMAT. */
                        SIOOUTBYTE      PROC NEAR
          0330  55              PUSH    BP
          0331  8BEC            MOV     BP,SP
                 PROCEDURE(B);
  52   2            DECLARE B BYTE;
  53   2            CALL SIO$OUT$CHAR(ASCII(SHR(B,4) AND 0FH));
                                                  ; STATEMENT # 53
          0333  8A5E04          MOV     BL,[BP].B
          0336  B104            MOV     CL,4H
          0338  D2EB            SHR     BL,CL
          033A  80E30F          AND     BL,0FH
          033D  B700            MOV     BH,0H
          033F  2EFF773E        PUSH    CS:ASCII[BX]
          0343  E8A9FF          CALL    SIOOUTCHAR
  54   2            CALL SIO$OUT$CHAR(ASCII(B AND 0FH));
                                                  ; STATEMENT # 54
          0346  8A5E04          MOV     BL,[BP].B
          0349  80E30F          AND     BL,0FH
          034C  B700            MOV     BH,0H
          034E  2EFF773E        PUSH    CS:ASCII[BX]
          0352  E89AFF          CALL    SIOOUTCHAR
  55   2            CHECK$SUM = CHECK$SUM - B;
                                                  ; STATEMENT # 55
          0355  8A4604          MOV     AL,[BP].B
          0358  28066700        SUB     CHECKSUM,AL
  56   2             END;
                                                  ; STATEMENT # 56
          035C  5D              POP     BP
          035D  C20200          RET     2H
                        SIOOUTBYTE      ENDP

  57   1       SIO$OUT$WORD:
                                                  ; STATEMENT # 57
                  /* THIS ROUTINE OUTPUTS THE INPUT PARAMETER AS 4 ASCII HEXADECIMAL
                     CHARACTERS TO THE USART OUTPUT PORT. */
                        SIOOUTWORD      PROC NEAR
          0360  55              PUSH    BP
          0361  8BEC            MOV     BP,SP
                 PROCEDURE(W);
  58   2            DECLARE W WORD;
  59   2            CALL SIO$OUT$BYTE(HIGH(W));
                                                  ; STATEMENT # 59
          0363  8B4604          MOV     AX,[BP].W
          0366  88E0            MOV     AL,AH
          0368  50              PUSH    AX      ; 1
          0369  E8C4FF          CALL    SIOOUTBYTE
  60   2            CALL SIO$OUT$BYTE(LOW(W));
                                                  ; STATEMENT # 60
          036C  8B4604          MOV     AX,[BP].W
          036F  50              PUSH    AX      ; 1
          0370  E8BDFF          CALL    SIOOUTBYTE
  61   2             END;
                                                  ; STATEMENT # 61
          0373  5D              POP     BP
          0374  C20200          RET     2H
                        SIOOUTWORD      ENDP
```

```
62   1       SIO$OUT$BLANK:
                                                ; STATEMENT # 62
              /* THIS ROUTINE OUTPUTS ONE BLANK. */
                    SIOOUTBLANK     PROC NEAR
0377  55              PUSH    BP
0378  8BEC            MOV     BP,SP
              PROCEDURE;
63   2                CALL SIO$OUT$CHAR(ASBL);
                                                ; STATEMENT # 63
037A  B020            MOV     AL,20H
037C  50              PUSH    AX      ; 1
037D  E86FFF          CALL    SIOOUTCHAR
64   2            END;
                                                ; STATEMENT # 64
0380  5D              POP     BP
0381  C3              RET
                    SIOOUTBLANK     ENDP

65   1       SIO$OUT$STRING:
                                                ; STATEMENT # 65
              /* OUTPUTS A STRING POINTED TO BY THE FIRST PARM. */
                    SIOOUTSTRING      PROC NEAR
0382  55              PUSH    BP
0383  8BEC            MOV     BP,SP
              PROCEDURE(PTR);
66   2                DECLARE PTR POINTER, STR BASED PTR (1) BYTE;
67   2                I = 0;
                                                ; STATEMENT # 67
0385  C606680000      MOV     I,0H
68   2                DO WHILE STR(I)<>0;
                                                ; STATEMENT # 68
                      @127:
038A  A06800          MOV     AL,I
038D  B400            MOV     AH,0H
038F  89C6            MOV     SI,AX
0391  C45E04          LES     BX,[BP].PTR
0394  26823800        CMP     ES:[BX].STR[SI],0H
0398  7417            JZ      @128
69   3                    CALL SIO$OUT$CHAR(STR(I));
                                                ; STATEMENT # 69
039A  A06800          MOV     AL,I
039D  B400            MOV     AH,0H
039F  89C6            MOV     SI,AX
03A1  C45E04          LES     BX,[BP].PTR
03A4  26FF30          PUSH    ES:[BX].STR[SI]
03A7  E845FF          CALL    SIOOUTCHAR
70   3                I = I + 1;
                                                ; STATEMENT # 70
03AA  8006680001      ADD     I,1H
71   3                END;
                                                ; STATEMENT # 71
03AF  EBD9            JMP     @127
                      @128:
72   2            END;
                                                ; STATEMENT # 72
03B1  5D              POP     BP
```

```
          03B2  C20400          RET     4H
                      SIOOUTSTRING      ENDP

73   1        SIO$OUT$HEADER:
                                                ; STATEMENT # 73
                 /* THIS ROUTINE OUTPUTS THE PAPER TAPE HEADER CONSISTING OF ':'
                    FOLLOWED BY THE RECORD LENGTH, LOAD ADDRESS, AND THE RECORD TYPE.
                    IT INITIALIZES THE CHECKSUM TO ZERO. */
                        SIOOUTHEADER        PROC NEAR
          03B5  55              PUSH    BP
          03B6  8BEC            MOV     BP,SP
                 PROCEDURE(LENGTH,LOAD$ADDR,REC$TYPE);
74   2              DECLARE (LENGTH,REC$TYPE) BYTE, LOAD$ADDR WORD;
75   2              CALL SIO$OUT$CHAR(':');
                                                ; STATEMENT # 75
          03B8  B03A            MOV     AL,3AH
          03BA  50              PUSH    AX      ; 1
          03BB  E831FF          CALL    SIOOUTCHAR
76   2              CHECK$SUM = 0;
                                                ; STATEMENT # 76
          03BE  C606670000      MOV     CHECKSUM,0H
77   2              CALL SIO$OUT$BYTE(LENGTH);
                                                ; STATEMENT # 77
          03C3  FF7608          PUSH    [BP].LENGTH; 1
          03C6  E867FF          CALL    SIOOUTBYTE
78   2              CALL SIO$OUT$WORD(LOAD$ADDR);
                                                ; STATEMENT # 78
          03C9  FF7606          PUSH    [BP].LOADADDR; 1
          03CC  E891FF          CALL    SIOOUTWORD
79   2              CALL SIO$OUT$BYTE(REC$TYPE);
                                                ; STATEMENT # 79
          03CF  FF7604          PUSH    [BP].RECTYPE; 1
          03D2  E85BFF          CALL    SIOOUTBYTE
80   2           END;
                                                ; STATEMENT # 80
          03D5  5D              POP     BP
          03D6  C20600          RET     6H
                      SIOOUTHEADER        ENDP

81   1        SIO$8251$SETTLING$DELAY:
                                                ; STATEMENT # 81
                 /* THIS DELAY ROUTINE ALLOWS THE USART SUFFICIENT TIME
                    TO SETTLE AFTER WRITE OPERATIONS. */
                        SIO8251SETTLINGDELAY        PROC NEAR
          03D9  55              PUSH    BP
          03DA  8BEC            MOV     BP,SP
                 PROCEDURE;
82   2              I = SHR(0FFH,0FFH);
                                                ; STATEMENT # 82
          03DC  B0FF            MOV     AL,0FFH
          03DE  B1FF            MOV     CL,0FFH
          03E0  D2E8            SHR     AL,CL
          03E2  A26800          MOV     I,AL
83   2           END;
                                                ; STATEMENT # 83
          03E5  5D              POP     BP
          03E6  C3              RET
```

```
                    SIO8251SETTLINGDELAY    ENDP


                /*******************************
                *   UTILITY ROUTINES SECTION   *
                *******************************/

84   1      SIO$VALID$HEX:
                                                ; STATEMENT # 84
              /* THIS ROUTINE TESTS IF THE INPUT PARM IS A VALID ASCII HEX DIGIT
                 AND RETURNS TRUE AS THE VALUE OF THE PROCEDURE IF SO AND FALSE
                 IF NOT. */
                   SIOVALIDHEX      PROC NEAR
     03E7  55              PUSH    BP
     03E8  8BEC            MOV     BP,SP
               PROCEDURE (H) BYTE;
85   2            DECLARE H BYTE;
86   2            DO I=0 TO LAST(ASCII);
                                                ; STATEMENT # 86
     03EA  C606680000      MOV     I,0H
                   @129:
     03EF  803E68000F      CMP     I,0FH
     03F4  771A            JA      @130
87   3                IF H=ASCII(I) THEN RETURN TRUE;
                                                ; STATEMENT # 87
     03F6  8A4604          MOV     AL,[BP].H
     03F9  8A1E6800        MOV     BL,I
     03FD  B700            MOV     BH,0H
     03FF  2E3A473E        CMP     AL,CS:ASCII[BX]
     0403  7504            JNZ     @9
                                                ; STATEMENT # 88
     0405  B0FF            MOV     AL,0FFH
     0407  EB09            JMP     @3
                   @9:
89   3            END;
                                                ; STATEMENT # 89
     0409  8006680001      ADD     I,1H
     040E  75DF            JNZ     @129
                   @130:
90   2            RETURN FALSE;
                                                ; STATEMENT # 90
     0410  B000            MOV     AL,0H
                   @3:
     0412  5D              POP     BP
     0413  C20200          RET     2H
91   2          END;
                                                ; STATEMENT # 91
                   SIOVALIDHEX      ENDP

92   1      SIO$HEX:
                                                ; STATEMENT # 92
             /* THIS ROUTINE CONVERTS THE INPUT PARM FROM ASCII TO ITS BINARY
                EQUIVALENT AND RETURNS IT AS THE VALUE OF THE PROCEDURE. NO CHECK
                IS MADE FOR INPUT VALIDITY. */
                   SIOHEX       PROC NEAR
     0416  55              PUSH    BP
     0417  8BEC            MOV     BP,SP
```

```
                PROCEDURE(C) WORD;
 93   2             DECLARE C BYTE;
 94   2             IF C<='9' THEN RETURN DOUBLE(C-30H);
                                                    ; STATEMENT # 94
          0419  807E0439        CMP     [BP].C,39H
          041D  7707            JA      @10
                                                    ; STATEMENT # 95
          041F  8A4604          MOV     AL,[BP].C
          0422  2C30            SUB     AL,30H
          0424  EB05            JMP     @4
                        @10:
 96   2             ELSE RETURN DOUBLE(C-37H);
                                                    ; STATEMENT # 96
          0426  8A4604          MOV     AL,[BP].C
          0429  2C37            SUB     AL,37H
                        @4:
          042B  B400            MOV     AH,0H
          042D  5D              POP     BP
          042E  C20200          RET     2H
 97   2         END;
                        SIOHEX      ENDP

 98   1       SIO$VALID$REG$FIRST:
                                                    ; STATEMENT # 98
                 /* THIS ROUTINE CHECKS IF 'CHAR' IS A VALID FIRST LETTER OF A REGISTER
                    NAME AND RETURNS TRUE AS THE VALUE OF THE PROCEDURE IF SO. */
                        SIOVALIDREGFIRST      PROC NEAR
          0431  55              PUSH    BP
          0432  8BEC            MOV     BP,SP
                  PROCEDURE BYTE;
 99   2             DO I=0 TO 26 BY 2;
                                                    ; STATEMENT # 99
          0434  C606680000      MOV     I,0H
          0439  EB07            JMP     @133
                        @131:
          043B  8006680002      ADD     I,2H
          0440  721A            JB      @132
                        @133:
          0442  803E68001A      CMP     I,1AH
          0447  7713            JA      @132
100   3                 IF CHAR=REG(I) THEN RETURN TRUE;
                                                    ; STATEMENT # 100
          0449  A06600          MOV     AL,CHAR
          044C  8A1E6800        MOV     BL,I
          0450  B700            MOV     BH,0H
          0452  2E3A4758        CMP     AL,CS:REG[BX]
          0456  75E3            JNZ     @131
                                                    ; STATEMENT # 101
          0458  B0FF            MOV     AL,0FFH
          045A  5D              POP     BP
          045B  C3              RET
102   3             END;
                                                    ; STATEMENT # 102
                        @132:
103   2             RETURN FALSE;
                                                    ; STATEMENT # 103
          045C  B000            MOV     AL,0H
```

```
          045E  5D              POP     BP
          045F  C3              RET
104   2        END;
                                        ; STATEMENT # 104
                    SIOVALIDREGFIRST    ENDP

105   1      SIO$VALID$REG:
                                        ; STATEMENT # 105
               /* THIS ROUTINE CHECKS IF THE TWO INPUT PARMS TAKEN TOGETHER FORM
                  A VALID REGISTER NAME.  IT SEARCHES THE REGISTER TABLE AND IF A
                  MATCH IS FOUND, THE GLOBAL 'REG$INDEX' IS SET TO THE INDEX OF THE
                  VALID REGISTER AND THE PROCEDURE RETURNS TRUE.  IF NO MATCH THE
                  PROCEDURE RETURNS FALSE AND REG$INDEX IS UNDEFINED. */
                   SIOVALIDREG     PROC NEAR
          0460  55              PUSH    BP
          0461  8BEC            MOV     BP,SP
               PROCEDURE (C1,C2) BYTE;
106   2           DECLARE (C1,C2) BYTE;
107   2           DO REG$INDEX=0 TO 13;
                                        ; STATEMENT # 107
          0463  C7062E000000    MOV     REGINDEX,0H
                    @134:
          0469  833E2E000D      CMP     REGINDEX,0DH
          046E  7731            JA      @135
108   3                 IF C1=REG(REG$INDEX*2) AND C2=REG(REG$INDEX*2+1) THEN
                                        ; STATEMENT # 108
          0470  8B1E2E00        MOV     BX,REGINDEX
          0474  D1E3            SHL     BX,1
          0476  8A4606          MOV     AL,[BP].C1
          0479  2E3A4758        CMP     AL,CS:REG[BX]
          047D  B0FF            MOV     AL,0FFH
          047F  7401            JZ      $+3H
          0481  40              INC     AX
          0482  8A4E04          MOV     CL,[BP].C2
          0485  50              PUSH    AX      ; 1
          0486  2E3A4F59        CMP     CL,CS:REG[BX+1H]
          048A  B0FF            MOV     AL,0FFH
          048C  7401            JZ      $+3H
          048E  40              INC     AX
          048F  59              POP     CX      ; 1
          0490  22C1            AND     AL,CL
          0492  D0D8            RCR     AL,1
          0494  7304            JNB     @13
109   3                   RETURN TRUE;
                                        ; STATEMENT # 109
          0496  B0FF            MOV     AL,0FFH
          0498  EB09            JMP     @6
                    @13:
110   3             END;
                                        ; STATEMENT # 110
          049A  83062E0001      ADD     REGINDEX,1H
          049F  73C8            JNB     @134
                    @135:
111   2           RETURN FALSE;
                                        ; STATEMENT # 111
          04A1  B000            MOV     AL,0H
                    @6:
```

```
          04A3  5D              POP     BP
          04A4  C20400          RET     4H
112   2        END;
                                        ; STATEMENT # 112
                SIOVALIDREG     ENDP

113   1     SIO$CRLF:
                                        ; STATEMENT # 113
            /* THIS ROUTINE OUTPUTS A CR AND LF TO THE OUTPUT PORT. */
                SIOCRLF     PROC NEAR
          04A7  55              PUSH    BP
          04A8  8BEC            MOV     BP,SP
            PROCEDURE;
114   2        CALL SIO$OUT$CHAR(ASCR);
                                        ; STATEMENT # 114
          04AA  B00D            MOV     AL,0DH
          04AC  50              PUSH    AX      ; 1
          04AD  E83FFE          CALL    SIOOUTCHAR
115   2        CALL SIO$OUT$CHAR(ASLF);
                                        ; STATEMENT # 115
          04B0  B00A            MOV     AL,0AH
          04B2  50              PUSH    AX      ; 1
          04B3  E839FE          CALL    SIOOUTCHAR
116   2        END;
                                        ; STATEMENT # 116
          04B6  5D              POP     BP
          04B7  C3              RET
                SIOCRLF     ENDP

117   1     SIO$TEST$WORD$MODE:
                                        ; STATEMENT # 117
            /* THIS PROCEDURE TESTS FOR A 'W' FOLLOWING THE COMMAND AND IF SO
               SETS THE FLAG 'WORD$MODE TO TRUE OR FALSE OTHERWISE. SCANS OFF
               OPTIONAL BLANK FOLLOWING COMMAND. */
                SIOTESTWORDMODE     PROC NEAR
          04B8  55              PUSH    BP
          04B9  8BEC            MOV     BP,SP
            PROCEDURE;
118   2        WORD$MODE = FALSE;
                                        ; STATEMENT # 118
          04BB  C606690000      MOV     WORDMODE,0H
119   2        CALL SIO$GET$CHAR;
                                        ; STATEMENT # 119
          04C0  E84CFE          CALL    SIOGETCHAR
120   2        IF CHAR='W' THEN
                                        ; STATEMENT # 120
          04C3  803E660057      CMP     CHAR,57H
          04C8  7508            JNZ     @14
121   2              DO;
122   3              WORD$MODE = TRUE;
                                        ; STATEMENT # 122
          04CA  C6066900FF      MOV     WORDMODE,0FFH
123   3              CALL SIO$GET$CHAR;
                                        ; STATEMENT # 123
          04CF  E83DFE          CALL    SIOGETCHAR
124   3              END;
                     @14:
```

```
125   2             IF CHAR=ASBL THEN
                                                ; STATEMENT # 125
          04D2  803E660020      CMP     CHAR,20H
          04D7  7503            JNZ     @15
126   2                 CALL SIO$GET$CHAR;
                                                ; STATEMENT # 126
          04D9  E833FE          CALL    SIOGETCHAR
                      @15:
127   2           END;
                                                ; STATEMENT # 127
          04DC  5D              POP     BP
          04DD  C3              RET
                      SIOTESTWORDMODE       ENDP

128   1       SIO$SCAN$BLANK:
                                                ; STATEMENT # 128
              /* THIS ROUTINE IS CALLED AFTER A COMMAND LETTER TO SCAN OFF THE
                 OPTIONAL BLANK. */
                      SIOSCANBLANK      PROC NEAR
          04DE  55              PUSH    BP
          04DF  8BEC            MOV     BP,SP
              PROCEDURE;
129   2             CALL SIO$GET$CHAR;
                                                ; STATEMENT # 129
          04E1  E82BFE          CALL    SIOGETCHAR
130   2             IF CHAR=ASBL THEN
                                                ; STATEMENT # 130
          04E4  803E660020      CMP     CHAR,20H
          04E9  7503            JNZ     @16
131   2                 CALL SIO$GET$CHAR;
                                                ; STATEMENT # 131
          04EB  E821FE          CALL    SIOGETCHAR
                      @16:
132   2           END;
                                                ; STATEMENT # 132
          04EE  5D              POP     BP
          04EF  C3              RET
                      SIOSCANBLANK      ENDP


              /*********************************************
              *   ARGUMENT EXPRESSION EVALUATOR SECTION    *
              *********************************************/

133   1       SIO$GET$WORD:
                                                ; STATEMENT # 133
              /* THIS ROUTINE READS CHARS FROM THE INPUT PORT AND EVALUATES
                 AN EXPRESSION CONSISTING OF '+-' AND OPERANDS OF HEX NUMBERS
                 AND REGISTER NAMES. */
                      SIOGETWORD        PROC NEAR
          04F0  55              PUSH    BP
          04F1  8BEC            MOV     BP,SP
              PROCEDURE WORD;
134   2             DECLARE (SAVE,W) WORD, (OPER,T) BYTE;
135   2             OPER = '+';
                                                ; STATEMENT # 135
          04F3  C6066B002B      MOV     OPER,2BH
```

```
136   2             W = 0;
                                                ; STATEMENT # 136
          04F8  C7064E000000    MOV     W,0H
137   2             DO WHILE TRUE;
                                                ; STATEMENT # 137
                    @136:
138   3             T = CHAR;
                                                ; STATEMENT # 138
          04FE  A06600          MOV     AL,CHAR
          0501  A26C00          MOV     T,AL
139   3             SAVE = 0;
                                                ; STATEMENT # 139
          0504  C7064C000000    MOV     SAVE,0H
140   3             IF SIO$VALID$REG$FIRST THEN
                                                ; STATEMENT # 140
          050A  E824FF          CALL    SIOVALIDREGFIRST
          050D  D0D8            RCR     AL,1
          050F  733B            JNB     @17
141   3                 DO;
142   4                 CALL SIO$GET$CHAR;
                                                ; STATEMENT # 142
          0511  E8FBFD          CALL    SIOGETCHAR
143   4                 IF SIO$VALID$REG(T,CHAR) THEN
                                                ; STATEMENT # 143
          0514  FF366C00        PUSH    T       ; 1
          0518  FF366600        PUSH    CHAR    ; 2
          051C  E841FF          CALL    SIOVALIDREG
          051F  D0D8            RCR     AL,1
          0521  731F            JNB     @18
144   4                     DO;
145   5                     SAVE = REG$SAV(REG$INDEX);
                                                ; STATEMENT # 145
          0523  8B1E2E00        MOV     BX,REGINDEX
          0527  D1E3            SHL     BX,1
          0529  8B4730          MOV     AX,REGSAV[BX]
          052C  A34C00          MOV     SAVE,AX
146   5                     CALL SIO$GET$CHAR;
                                                ; STATEMENT # 146
          052F  E8DDFD          CALL    SIOGETCHAR
147   5                     GOTO EVAL;
                                                ; STATEMENT # 147
                      @7:
          0532  803E6B002B      CMP     OPER,2BH
          0537  7548            JNZ     EVAL
          0539  A14C00          MOV     AX,SAVE
          053C  01064E00        ADD     W,AX
          0540  EB46            JMP     @22
148   5                     END;
                                                ; STATEMENT # 148
                    @18:
                      ELSE
149   4                     SAVE = SIO$HEX(T);
                                                ; STATEMENT # 149
          0542  FF366C00        PUSH    T       ; 1
          0546  E8CDFE          CALL    SIOHEX
          0549  A34C00          MOV     SAVE,AX
150   4                 END;
```

```
                    @17:
151   3               IF NOT(SIO$VALID$HEX(T)) THEN  GOTO ERROR;
                                                    ; STATEMENT # 151
        054C  FF366C00          PUSH    T           ; 1
        0550  E894FE            CALL    SIOVALIDHEX
        0553  F6D0              NOT     AL
        0555  D0D8              RCR     AL,1
        0557  7303              JNB     $+5H
        0559  E9AAFC            JMP     ERROR
153   3               DO WHILE SIO$VALID$HEX(CHAR);
                                                    ; STATEMENT # 153
                    @138:
        055C  FF366600          PUSH    CHAR        ; 1
        0560  E884FE            CALL    SIOVALIDHEX
        0563  D0D8              RCR     AL,1
        0565  73CB              JNB     @7
154   4                   SAVE = SHL(SAVE,4) + SIO$HEX(CHAR);
                                                    ; STATEMENT # 154
        0567  A14C00            MOV     AX,SAVE
        056A  B104              MOV     CL,4H
        056C  D3E0              SHL     AX,CL
        056E  50                PUSH    AX          ; 1
        056F  FF366600          PUSH    CHAR        ; 2
        0573  E8A0FE            CALL    SIOHEX
        0576  59                POP     CX          ; 1
        0577  03C1              ADD     AX,CX
        0579  A34C00            MOV     SAVE,AX
155   4                   CALL SIO$GET$CHAR;
                                                    ; STATEMENT # 155
        057C  E890FD            CALL    SIOGETCHAR
156   4               END;
                                                    ; STATEMENT # 156
        057F  EBDB              JMP     @138
157   3           EVAL:   IF OPER='+' THEN
                                                    ; STATEMENT # 157
                    EVAL:
158   3                   W = W + SAVE;
                    ELSE
159   3                   W = W - SAVE;
                                                    ; STATEMENT # 159
        0581  A14C00            MOV     AX,SAVE
        0584  29064E00          SUB     W,AX
                    @22:
160   3               IF CHAR=ASCR OR CHAR=':' OR CHAR=',' THEN
                                                    ; STATEMENT # 160
        0588  A06600            MOV     AL,CHAR
        058B  3C0D              CMP     AL,0DH
        058D  B0FF              MOV     AL,0FFH
        058F  7401              JZ      $+3H
        0591  40                INC     AX
        0592  50                PUSH    AX          ; 1
        0593  803E66003A        CMP     CHAR,3AH
        0598  B0FF              MOV     AL,0FFH
        059A  7401              JZ      $+3H
        059C  40                INC     AX
        059D  59                POP     CX          ; 1
        059E  0AC1              OR      AL,CL
```

```
            05A0  50              PUSH    AX      ; 1
            05A1  803E66002C      CMP     CHAR,2CH
            05A6  B0FF            MOV     AL,0FFH
            05A8  7401            JZ      $+3H
            05AA  40              INC     AX
            05AB  59              POP     CX      ; 1
            05AC  0AC1            OR      AL,CL
            05AE  D0D8            RCR     AL,1
            05B0  7305            JNB     @23
161   3                   RETURN W;
                                                  ; STATEMENT # 161
            05B2  A14E00          MOV     AX,W
            05B5  5D              POP     BP
            05B6  C3              RET
                        @23:
162   3               IF CHAR='+' OR CHAR='-' THEN
                                                  ; STATEMENT # 162
            05B7  803E66002B      CMP     CHAR,2BH
            05BC  B0FF            MOV     AL,0FFH
            05BE  7401            JZ      $+3H
            05C0  40              INC     AX
            05C1  50              PUSH    AX      ; 1
            05C2  803E66002D      CMP     CHAR,2DH
            05C7  B0FF            MOV     AL,0FFH
            05C9  7401            JZ      $+3H
            05CB  40              INC     AX
            05CC  59              POP     CX      ; 1
            05CD  0AC1            OR      AL,CL
            05CF  D0D8            RCR     AL,1
            05D1  7203            JB      $+5H
            05D3  E930FC          JMP     ERROR
163   3                   OPER = CHAR;
                                                  ; STATEMENT # 163
            05D6  A06600          MOV     AL,CHAR
            05D9  A26B00          MOV     OPER,AL
                        ELSE
164   3                   GOTO ERROR;
165   3               CALL SIO$GET$CHAR;
                                                  ; STATEMENT # 165
            05DC  E830FD          CALL    SIOGETCHAR
166   3           END;
                                                  ; STATEMENT # 166
            05DF  E91CFF          JMP     @136
                        @137:
167   2        END;
                                                  ; STATEMENT # 167
            05E2  5D              POP     BP
            05E3  C3              RET
                        SIOGETWORD      ENDP

168   1       SIO$GET$ADDR:
                                                  ; STATEMENT # 168
                /* THIS ROUTINE ACCEPTS A VALID ADDRESS EXPRESSION CONSISTING
                   OF AN OPTIONAL <SEG>: AND AN DISPLACEMENT. */
                    SIOGETADDR      PROC NEAR
            05E4  55              PUSH    BP
            05E5  8BEC            MOV     BP,SP
```

```
                PROCEDURE(PTR,DEFAULT$BASE);
169   2            DECLARE PTR POINTER, DEFAULT$BASE WORD,
                      ARG BASED PTR STRUCTURE (OFF WORD, SEG WORD);
170   2            ARG.SEG = DEFAULT$BASE;
                                                  ; STATEMENT # 170
          05E7  8B4604          MOV     AX,[BP].DEFAULTBASE
          05EA  C45E06          LES     BX,[BP].PTR
          05ED  26894702        MOV     ES:ARG[BX+2H],AX
171   2            ARG.OFF = SIO$GET$WORD;
                                                  ; STATEMENT # 171
          05F1  E8FCFE          CALL    SIOGETWORD
          05F4  C45E06          LES     BX,[BP].PTR
          05F7  268907          MOV     ES:ARG[BX],AX
172   2            IF CHAR=':' THEN
                                                  ; STATEMENT # 172
          05FA  803E66003A      CMP     CHAR,3AH
          05FF  7520            JNZ     @26
173   2                DO;
174   3                CALL SIO$GET$CHAR;
                                                  ; STATEMENT # 174
          0601  E80BFD          CALL    SIOGETCHAR
175   3                ARG.SEG = ARG.OFF;
                                                  ; STATEMENT # 175
          0604  C45E06          LES     BX,[BP].PTR
          0607  268B07          MOV     AX,ES:ARG[BX]
          060A  26894702        MOV     ES:ARG[BX+2H],AX
176   3                ARG.OFF = SIO$GET$WORD;
                                                  ; STATEMENT # 176
          060E  E8DFFE          CALL    SIOGETWORD
          0611  C45E06          LES     BX,[BP].PTR
          0614  268907          MOV     ES:ARG[BX],AX
177   3                IF CHAR=':' THEN  GOTO ERROR;
                                                  ; STATEMENT # 177
          0617  803E66003A      CMP     CHAR,3AH
          061C  7503            JNZ     $+5H
          061E  E9E5FB          JMP     ERROR
                                                  ; STATEMENT # 178
179   3                END;
                        @26:
180   2            END;
                                                  ; STATEMENT # 180
          0621  5D              POP     BP
          0622  C20600          RET     6H
                SIOGETADDR      ENDP

181   1       SIO$UPDATE$IP:
                                                  ; STATEMENT # 181
                /* THIS PROCEDURE IS CALLED BY SINGLE STEP AND GO TO OUTPUT THE CURRENT
                   IP AND INSTRUCTION BYTE AND OPEN THE IP FOR INPUT. */
                    SIOUPDATEIP     PROC NEAR
          0625  55              PUSH    BP
          0626  8BEC            MOV     BP,SP
                PROCEDURE;
182   2           CALL SIO$OUT$BLANK;
                                                  ; STATEMENT # 182
          0628  E84CFD          CALL    SIOOUTBLANK
183   2           CALL SIO$OUT$WORD(IP);
```

```
                                                    ; STATEMENT # 183
        062B  FF364800          PUSH    REGSAV+18H; 1
        062F  E82EFD            CALL    SIOOUTWORD
184  2          CSIP.SEG = CS;
                                                    ; STATEMENT # 184
        0632  A14000            MOV     AX,REGSAV+10H
        0635  A32800            MOV     CSIP+2H,AX
185  2          CSIP.OFF = IP;
                                                    ; STATEMENT # 185
        0638  A14800            MOV     AX,REGSAV+18H
        063B  A32600            MOV     CSIP,AX
186  2          CALL SIO$OUT$CHAR('-');
                                                    ; STATEMENT # 186
        063E  B02D              MOV     AL,2DH
        0640  50                PUSH    AX      ; 1
        0641  E8ABFC            CALL    SIOOUTCHAR
187  2          CALL SIO$OUT$BLANK;
                                                    ; STATEMENT # 187
        0644  E830FD            CALL    SIOOUTBLANK
188  2          CALL SIO$OUT$BYTE(MEMORY$CSIP);
                                                    ; STATEMENT # 188
        0647  C41E2600          LES     BX,MEMORYCSIPPTR
        064B  26FF37            PUSH    ES:MEMORYCSIP[BX]
        064E  E8DFFC            CALL    SIOOUTBYTE
189  2          CALL SIO$OUT$BLANK;
                                                    ; STATEMENT # 189
        0651  E823FD            CALL    SIOOUTBLANK
190  2          CALL SIO$GET$CHAR;
                                                    ; STATEMENT # 190
        0654  E8B8FC            CALL    SIOGETCHAR
191  2          IF CHAR<>',' AND CHAR<>ASCR THEN CALL SIO$GET$ADDR(@CSIP,CS);
                                                    ; STATEMENT # 191
        0657  803E66002C        CMP     CHAR,2CH
        065C  B0FF              MOV     AL,0FFH
        065E  7501              JNZ     $+3H
        0660  40                INC     AX
        0661  50                PUSH    AX      ; 1
        0662  803E66000D        CMP     CHAR,0DH
        0667  B0FF              MOV     AL,0FFH
        0669  7501              JNZ     $+3H
        066B  40                INC     AX
        066C  59                POP     CX      ; 1
        066D  22C1              AND     AL,CL
        066F  D0D8              RCR     AL,1
        0671  730D              JNB     @28
                                                    ; STATEMENT # 192
        0673  8D062600          LEA     AX,CSIP
        0677  1E                PUSH    DS      ; 1
        0678  50                PUSH    AX      ; 2
        0679  FF364000          PUSH    REGSAV+10H; 3
        067D  E864FF            CALL    SIOGETADDR
                      @28:
193  2           END;
                                                    ; STATEMENT # 193
        0680  5D                POP     BP
        0681  C3                RET
                    SIOUPDATEIP     ENDP
```

```
                /*******************************
                *   PAPER TAPE READ SECTION   *
                *******************************/

194   1       SIO$READ$CHAR:
                                                ; STATEMENT # 194
                /* THIS PROCEDURE READS A BYTE FROM THE PAPER TAPE READER OF THE TTY
                   BY ACTIVATING DTR, SAMPLING DSR FOR THE START BIT, DEACTIVATING
                   DTR WHEN THE START BIT APPEARS, AND READING THE DATUM FROM THE
                   USART. TO PROVIDE A HOLD-OFF CAPABILITY FOR THE LOAD COMMAND
                   ONLY (NOT INTENDED FOR TTY PAPER TAPE) CONTROL-S WILL CAUSE THIS
                   ROUTINE TO WAIT FOR A CORRESPONDING CONTROL-Q TO BE READ BEFORE
                   CONTINUING. */
                    SIOREADCHAR      PROC NEAR
        0682  55              PUSH    BP
        0683  8BEC            MOV     BP,SP
                PROCEDURE BYTE;
195   2           DECLARE DELAY WORD;
196   2       LOOP:
                                                ; STATEMENT # 196
                  LOOP:
                OUTPUT(SIO$STAT$PORT) = SIO$8251$DTR$ON;        /* DTR ON */
        0685  BAF2FF          MOV     DX,0FFF2H
        0688  B027            MOV     AL,27H
        068A  EE              OUT     DX
197   2           DELAY = 0;
                                                ; STATEMENT # 197
        068B  C706500000000   MOV     DELAY,0H
198   2           DO WHILE (INPUT(SIO$STAT$PORT) AND SIO$DSRDY)=0; /*WAIT STARTBIT*/
                                                ; STATEMENT # 198
                  @140:
        0691  BAF2FF          MOV     DX,0FFF2H
        0694  EC              IN      DX
        0695  A880            TEST    AL,80H
        0697  7517            JNZ     @141
199   3               DELAY = DELAY + 1;
                                                ; STATEMENT # 199
        0699  A15000          MOV     AX,DELAY
        069C  83C001          ADD     AX,1H
        069F  A35000          MOV     DELAY,AX
200   3               IF DELAY>=MAX$DELAY THEN
                                                ; STATEMENT # 200
        06A2  3D1027          CMP     AX,2710H
        06A5  72EA            JB      @140
201   3                 DO;
202   4                 OUTPUT(SIO$STAT$PORT) = SIO$8251$CMND;     /* DTR OFF */
                                                ; STATEMENT # 202
        06A7  BAF2FF          MOV     DX,0FFF2H
        06AA  B025            MOV     AL,25H
        06AC  EE              OUT     DX
203   4                 GOTO ERROR;
                                                ; STATEMENT # 203
        06AD  E956FB          JMP     ERROR
204   4                 END;
205   3           END;
```

```
                                                ; STATEMENT # 205
                    @141:
206   2               OUTPUT(SIO$STAT$PORT) = SIO$8251$CMND;          /* DTR OFF */
                                                ; STATEMENT # 206
          06B0  BAF2FF          MOV     DX,0FFF2H
          06B3  B025            MOV     AL,25H
          06B5  EE              OUT     DX
207   2               DO WHILE (INPUT(SIO$STAT$PORT) AND SIO$RXRDY)=0;END;  /* WAIT LOOP */
                                                ; STATEMENT # 207
                    @142:
          06B6  BAF2FF          MOV     DX,0FFF2H
          06B9  EC              IN      DX
          06BA  A802            TEST    AL,2H
          06BC  74F8            JZ      @142
209   2               CALL SIO$CHECK$CONTROL$CHAR;
                                                ; STATEMENT # 209
          06BE  E8F2FB          CALL    SIOCHECKCONTROLCHAR
210   2               IF CHAR = 11H THEN GOTO LOOP;  /* GET ANOTHER IF CTL-Q */
                                                ; STATEMENT # 210
          06C1  803E660011      CMP     CHAR,11H
          06C6  74BD            JZ      LOOP
212   2               RETURN CHAR;
                                                ; STATEMENT # 212
          06C8  A06600          MOV     AL,CHAR
          06CB  5D              POP     BP
          06CC  C3              RET
213   2            END;
                                                ; STATEMENT # 213
                    SIOREADCHAR     ENDP

214   1          SIO$READ$BYTE:
                                                ; STATEMENT # 214
                  /* THIS ROUTINE READS A BYTE FROM THE PAPER TAPE READER. */
                    SIOREADBYTE     PROC NEAR
          06CD  55              PUSH    BP
          06CE  8BEC            MOV     BP,SP
                  PROCEDURE BYTE;
215   2               DECLARE T BYTE;
216   2               T = LOW(SIO$HEX(SIO$READ$CHAR));
                                                ; STATEMENT # 216
          06D0  E8AFFF          CALL    SIOREADCHAR
          06D3  50              PUSH    AX      ; 1
          06D4  E83FFD          CALL    SIOHEX
          06D7  A26D00          MOV     T,AL
217   2               T = SHL(T,4) + LOW(SIO$HEX(SIO$READ$CHAR));
                                                ; STATEMENT # 217
          06DA  A06D00          MOV     AL,T
          06DD  B104            MOV     CL,4H
          06DF  D2E0            SHL     AL,CL
          06E1  50              PUSH    AX      ; 1
          06E2  E89DFF          CALL    SIOREADCHAR
          06E5  50              PUSH    AX      ; 2
          06E6  E82DFD          CALL    SIOHEX
          06E9  59              POP     CX      ; 1
          06EA  02C1            ADD     AL,CL
          06EC  A26D00          MOV     T,AL
218   2               CHECK$SUM = CHECK$SUM + T;
```

```
                                                ; STATEMENT # 218
        06EF  A06D00            MOV     AL,T
        06F2  00066700          ADD     CHECKSUM,AL
219  2          RETURN T;
                                                ; STATEMENT # 219
        06F6  5D                POP     BP
        06F7  C3                RET
220  2       END;
                                                ; STATEMENT # 220
                    SIOREADBYTE     ENDP

221  1      SIO$READ$WORD:
                                                ; STATEMENT # 221
             /* THIS ROUTINE READS A WORD FROM THE PAPER TAPE READER AND RETURNS IT
                AS THE VALUE OF THE PROCEDURE. */
                    SIOREADWORD     PROC NEAR
        06F8  55                PUSH    BP
        06F9  8BEC              MOV     BP,SP
             PROCEDURE WORD;
222  2          DECLARE T BYTE;
223  2          T = SIO$READ$BYTE;
                                                ; STATEMENT # 223
        06FB  E8CFFF            CALL    SIOREADBYTE
        06FE  A26E00            MOV     T,AL
224  2          RETURN SHL(DOUBLE(T),8) + DOUBLE(SIO$READ$BYTE);
                                                ; STATEMENT # 224
        0701  A06E00            MOV     AL,T
        0704  B400              MOV     AH,0H
        0706  B108              MOV     CL,8H
        0708  D3E0              SHL     AX,CL
        070A  50                PUSH    AX      ; 1
        070B  E8BFFF            CALL    SIOREADBYTE
        070E  B400              MOV     AH,0H
        0710  59                POP     CX      ; 1
        0711  03C1              ADD     AX,CX
        0713  5D                POP     BP
        0714  C3                RET
225  2      END;
                                                ; STATEMENT # 225
                    SIOREADWORD     ENDP


             /*********************************************
             *   INTERRUPT AND RESTORE/EXECUTE SECTION   *
             *********************************************/

226  1      SAVE$REGISTERS:
                                                ; STATEMENT # 226
             /* THIS ROUTINE IS USED TO SAVE THE STACKED USER'S REGISTERS IN THE
                MONITOR'S SAVE AREA. */
                    SAVEREGISTERS     PROC NEAR
        0715  55                PUSH    BP
        0716  8BEC              MOV     BP,SP
             PROCEDURE;
227  2          BP = MEMORY$USERSTACK;
                                                ; STATEMENT # 227
        0718  C41E2A00          LES     BX,MEMORYUSERSTACKPTR
```

```
          071C  268B07          MOV     AX,ES:MEMORYUSERSTACK[BX]
          071F  A33A00          MOV     REGSAV+0AH,AX
228   2             USERSTACK.OFF = USERSTACK.OFF + 4;
                                                ; STATEMENT # 228
          0722  83062A0004      ADD     USERSTACK,4H
229   2             DO I=0 TO 10;                     /* POP REGISTERS OFF OF STACK */
                                                ; STATEMENT # 229
          0727  C606680000      MOV     I,0H
                      @144:
          072C  803E68000A      CMP     I,0AH
          0731  7725            JA      @145
230   3                 REG$SAV(REG$ORD(I)) = MEMORY$USERSTACK;
                                                ; STATEMENT # 230
          0733  8A1E6800        MOV     BL,I
          0737  B700            MOV     BH,0H
          0739  2E8A9F7400      MOV     BL,CS:REGORD[BX]
          073E  B700            MOV     BH,0H
          0740  D1E3            SHL     BX,1
          0742  C4362A00        LES     SI,MEMORYUSERSTACKPTR
          0746  268B04          MOV     AX,ES:MEMORYUSERSTACK[SI]
          0749  894730          MOV     REGSAV[BX],AX
231   3                 USERSTACK.OFF = USERSTACK.OFF + 2;
                                                ; STATEMENT # 231
          074C  83062A0002      ADD     USERSTACK,2H
232   3             END;
                                                ; STATEMENT # 232
          0751  8006680001      ADD     I,1H
          0756  75D4            JNZ     @144
                      @145:
233   2             SS = USERSTACK.SEG;
                                                ; STATEMENT # 233
          0758  A12C00          MOV     AX,USERSTACK+2H
          075B  A34400          MOV     REGSAV+14H,AX
234   2             SP = USERSTACK.OFF;
                                                ; STATEMENT # 234
          075E  A12A00          MOV     AX,USERSTACK
          0761  A33800          MOV     REGSAV+8H,AX
235   2           END;
                                                ; STATEMENT # 235
          0764  5D              POP     BP
          0765  C3              RET
                    SAVEREGISTERS       ENDP

236   1       RESTORE$EXECUTE:
                                                ; STATEMENT # 236
                  /* THIS PROCEDURE RESTORES THE STATE OF THE USER MACHINE AND
                     PASSES CONTROL BACK TO THE USER PROGRAM. IT CONTAINS A
                     MACHINE LANGUAGE SUBROUTINE TO PERFORM THE POPPING OF THE
                     USER REGISTERS AND TO EXECUTE AN 'IRET' TO TRANSFER CONTROL
                     TO THE USER'S PROGRAM. */
                         RESTOREEXECUTE       PROC NEAR
          0766  55              PUSH    BP
          0767  8BEC            MOV     BP,SP
                 PROCEDURE;
237   2             DECLARE RESTORE$EXECUTE$CODE(*) BYTE DATA
                        (08BH,0ECH,           /* MOV BP,SP          */
                         08BH,046H,002H,      /* MOV AX,/BP/.PARM2  */
```

```
                        08BH,05EH,004H,      /* MOV BX,/BP/.PARM1   */
                        08EH,0D0H,           /* MOV SS,AX           */
                        08BH,0E3H,           /* MOV SP,BX           */
                        05DH,                /* POP BP              */
                        05FH,                /* POP DI              */
                        05EH,                /* POP SI              */
                        05BH,                /* POP BX              */
                        05AH,                /* POP DX              */
                        059H,                /* POP CX              */
                        058H,                /* POP AX              */
                        01FH,                /* POP DS              */
                        007H,                /* POP ES              */
                        0CFH),               /* IRET                */
                       RESTORE$EXECUTE$CODE$PTR WORD DATA (.RESTORE$EXECUTE$CODE);

238   2              USERSTACK.SEG = SS;
                                             ; STATEMENT # 238
          0769  A14400          MOV      AX,REGSAV+14H
          076C  A32C00          MOV      USERSTACK+2H,AX
239   2              USERSTACK.OFF = SP;
                                             ; STATEMENT # 239
          076F  A13800          MOV      AX,REGSAV+8H
          0772  A32A00          MOV      USERSTACK,AX
240   2              DO I=0 TO 10;            /* PUSH USER'S REGISTERS ONTO HIS STACK */
                                             ; STATEMENT # 240
          0775  C606680000      MOV      I,0H
                      @146:
          077A  803E68000A      CMP      I,0AH
          077F  7727            JA       @147
241   3                  USERSTACK.OFF = USERSTACK.OFF - 2;
                                             ; STATEMENT # 241
          0781  832E2A0002      SUB      USERSTACK,2H
242   3                  MEMORY$USERSTACK = REG$SAV(REG$ORD(10-I));
                                             ; STATEMENT # 242
          0786  B30A            MOV      BL,0AH
          0788  2A1E6800        SUB      BL,I
          078C  B700            MOV      BH,0H
          078E  2E8A9F7400      MOV      BL,CS:REGORD[BX]
          0793  B700            MOV      BH,0H
          0795  D1E3            SHL      BX,1
          0797  8B4730          MOV      AX,REGSAV[BX]
          079A  C41E2A00        LES      BX,MEMORYUSERSTACKPTR
          079E  268907          MOV      ES:MEMORYUSERSTACK[BX],AX
243   3              END;
                                             ; STATEMENT # 243
          07A1  8006680001      ADD      I,1H
          07A6  75D2            JNZ      @146
                      @147:
244   2              USERSTACK.OFF = USERSTACK.OFF - 2;
                                             ; STATEMENT # 244
          07A8  832E2A0002      SUB      USERSTACK,2H
245   2              MEMORY$USERSTACK = BP;
                                             ; STATEMENT # 245
          07AD  A13A00          MOV      AX,REGSAV+0AH
          07B0  C41E2A00        LES      BX,MEMORYUSERSTACKPTR
          07B4  268907          MOV      ES:MEMORYUSERSTACK[BX],AX
246   2              CALL RESTORE$EXECUTE$CODE$PTR(USERSTACK.OFF,USERSTACK.SEG);
```

```
                                                ; STATEMENT # 246
        07B7  FF362A00          PUSH    USERSTACK; 1
        07BB  FF362C00          PUSH    USERSTACK+2H; 2
        07BF  2EFF160400        CALL    CS:RESTOREEXECUTECODEPTR
247  2       END;
                                                ; STATEMENT # 247
        07C4  5D                POP     BP
        07C5  C3                RET
                RESTOREEXECUTE      ENDP


248  1      INTERRUPT1$ENTRY:
                                                ; STATEMENT # 248
          /* THIS PROCEDURE IS CALLED WHEN THE CPU IS INTERRUPTED BY EXECUTING
             AN INSTRUCTION WITH THE TRAP BIT SET (SINGLE STEP). */
        07C6  06                PUSH    ES
        07C7  1E                PUSH    DS
        07C8  2E8E1EA600        MOV     DS,CS:@@DATA$FRAME
        07CD  50                PUSH    AX
        07CE  51                PUSH    CX
        07CF  52                PUSH    DX
        07D0  53                PUSH    BX
        07D1  56                PUSH    SI
        07D2  57                PUSH    DI
        07D3  E80900            CALL    INTERRUPT1ENTRY
        07D6  5F                POP     DI
        07D7  5E                POP     SI
        07D8  5B                POP     BX
        07D9  5A                POP     DX
        07DA  59                POP     CX
        07DB  58                POP     AX
        07DC  1F                POP     DS
        07DD  07                POP     ES
        07DE  CF                IRET
              INTERRUPT1ENTRY       PROC NEAR
        07DF  55                PUSH    BP
        07E0  8BEC              MOV     BP,SP
          PROCEDURE INTERRUPT 1;
249  2       USERSTACK.OFF = STACKPTR;        /* CHANGE TO MONITOR'S STACK */
                                                ; STATEMENT # 249
        07E2  89E0              MOV     AX,SP
        07E4  A32A00            MOV     USERSTACK,AX
250  2       USERSTACK.SEG = STACKBASE;
                                                ; STATEMENT # 250
        07E7  8CD0              MOV     AX,SS
        07E9  A32C00            MOV     USERSTACK+2H,AX
251  2          STACKPTR =  MONITOR$STACKPTR;
                                                ; STATEMENT # 251
        07EC  A11400            MOV     AX,MONITORSTACKPTR
        07EF  89C4              MOV     SP,AX
252  2       STACKBASE = MONITOR$STACKBASE;
                                                ; STATEMENT # 252
        07F1  A11600            MOV     AX,MONITORSTACKBASE
        07F4  8ED0              MOV     SS,AX
253  2       CALL SAVE$REGISTERS;
                                                ; STATEMENT # 253
        07F6  E81CFF            CALL    SAVEREGISTERS
```

```
254   2             FL = FL AND (NOT STEP$TRAP);     /* CLEAR STEP FLAG */
                                                    ; STATEMENT # 254
        07F9  81264A00FFFE      AND     REGSAV+1AH,0FEFFH
255   2             IF LAST$COMMAND<>SS$COMMAND THEN  /* CONTINUE IF NOT SS */
                                                    ; STATEMENT # 255
        07FF  803E6A0003        CMP     LASTCOMMAND,3H
        0804  7403              JZ      @31
256   2                 CALL RESTORE$EXECUTE;
                                                    ; STATEMENT # 256
        0806  E85DFF            CALL    RESTOREEXECUTE
                    @31:
257   2             CALL SIO$CRLF;
                                                    ; STATEMENT # 257
        0809  E89BFC            CALL    SIOCRLF
258   2             CALL SIO$UPDATE$IP;
                                                    ; STATEMENT # 258
        080C  E816FE            CALL    SIOUPDATEIP
259   2             IF CHAR=',' THEN
                                                    ; STATEMENT # 259
        080F  803E66002C        CMP     CHAR,2CH
        0814  7515              JNZ     @32
260   2                 DO;
261   3                 IP = CSIP.OFF;
                                                    ; STATEMENT # 261
        0816  A12600            MOV     AX,CSIP
        0819  A34800            MOV     REGSAV+18H,AX
262   3                 CS = CSIP.SEG;
                                                    ; STATEMENT # 262
        081C  A12800            MOV     AX,CSIP+2H
        081F  A34000            MOV     REGSAV+10H,AX
263   3                   FL = FL OR STEP$TRAP;          /* SET STEP FLAG */
                                                    ; STATEMENT # 263
        0822  810E4A000001      OR      REGSAV+1AH,100H
264   3                 CALL RESTORE$EXECUTE;
                                                    ; STATEMENT # 264
        0828  E83BFF            CALL    RESTOREEXECUTE
265   3                 END;
                           @32:
266   2             IF CHAR<>ASCR THEN  GOTO ERROR;
                                                    ; STATEMENT # 266
        082B  803E66000D        CMP     CHAR,0DH
        0830  7403              JZ      $+5H
        0832  E9D1F9            JMP     ERROR
268   2             GOTO NEXT$COMMAND;
                                                    ; STATEMENT # 268
        0835  E937F9            JMP     NEXTCOMMAND
269   2           END;
                  INTERRUPT1ENTRY       ENDP

270   1       INTERRUPT3$ENTRY:
                                                    ; STATEMENT # 270
              /* THIS PROCEDURE IS CALLED WHEN THE CPU EXECUTES A 'INT 3' INSTRUCTION.
                 THE MONITOR INSERTS THIS (0CCH) FOR A BREAKPOINT.  ALSO AN EXTERNAL
                 INTERRUPT OR A USER SOFTWARE INTERRUPT MAY CAUSE THIS PROCEDURE TO BE
                 CALLED. */
        0838  06                PUSH    ES
        0839  1E                PUSH    DS
```

```
                083A  2E8E1EA600        MOV     DS,CS:@@DATA$FRAME
                083F  50                PUSH    AX
                0840  51                PUSH    CX
                0841  52                PUSH    DX
                0842  53                PUSH    BX
                0843  56                PUSH    SI
                0844  57                PUSH    DI
                0845  E80900            CALL    INTERRUPT3ENTRY
                0848  5F                POP     DI
                0849  5E                POP     SI
                084A  5B                POP     BX
                084B  5A                POP     DX
                084C  59                POP     CX
                084D  58                POP     AX
                084E  1F                POP     DS
                084F  07                POP     ES
                0850  CF                IRET
                          INTERRUPT3ENTRY       PROC NEAR
                0851  55                PUSH    BP
                0852  8BEC              MOV     BP,SP
                 PROCEDURE INTERRUPT 3;
271   2             USERSTACK.OFF = STACKPTR;
                                                ; STATEMENT # 271
                0854  89E0              MOV     AX,SP
                0856  A32A00            MOV     USERSTACK,AX
272   2             USERSTACK.SEG = STACKBASE;
                                                ; STATEMENT # 272
                0859  8CD0              MOV     AX,SS
                085B  A32C00            MOV     USERSTACK+2H,AX
273   2             STACKPTR =  MONITOR$STACKPTR;
                                                ; STATEMENT # 273
                085E  A11400            MOV     AX,MONITORSTACKPTR
                0861  89C4              MOV     SP,AX
274   2             STACKBASE = MONITOR$STACKBASE;
                                                ; STATEMENT # 274
                0863  A11600            MOV     AX,MONITORSTACKBASE
                0866  8ED0              MOV     SS,AX
275   2             CALL SAVE$REGISTERS;
                                                ; STATEMENT # 275
                0868  E8AAFE            CALL    SAVEREGISTERS
276   2             CALL SIO$CRLF;
                                                ; STATEMENT # 276
                086B  E839FC            CALL    SIOCRLF
277   2             GOTO AFTER$INTERRUPT;
                                                ; STATEMENT # 277
                086E  E9A7F9            JMP     AFTERINTERRUPT
278   2           END;
                          INTERRUPT3ENTRY       ENDP

279   1       INIT$INT$VECTOR:
                                                ; STATEMENT # 279
                  /* THIS ROUTINE INITIALIZES AN INTERRRUPT VECTOR AS FOLLOWS: THE OFFSET
                     FROM THE ADDRESS OF 'INT$ROUTINE' CORRECTED BY THE APPROPRIATE
                     NUMBER OF BYTES FOR THE INTERRUPT PLM PROLOGUE. THE SEGMENT FROM THE
                     CURRENT CS REGISTER IS DETERMINED BY A MACHINE LANGUAGE CODED
                     SUBROUTINE. */
                          INITINTVECTOR         PROC NEAR
```

```
          0871  55              PUSH    BP
          0872  8BEC            MOV     BP,SP
                 PROCEDURE(INT$VECTOR$PTR,INT$ROUTINE$OFFSET);
280   2           DECLARE INT$VECTOR$PTR POINTER, INT$ROUTINE$OFFSET WORD,
                     VECTOR BASED INT$VECTOR$PTR STRUCTURE (OFF WORD, SEG WORD),
                     CORRECTION LITERALLY '19H',  /* OFFSET FOR PROLOGUE */
                     INIT$INT$VECTOR$CODE(*) BYTE DATA
                     (055H,                          /* PUSH BP            */
                      08BH,0ECH,                     /* MOV BP,SP          */
                      08CH,0C8H,                     /* MOV AX,CS          */
                      0C4H,05EH,004H,                /* LES BX,/BP/.PARM1  */
                      026H,089H,007H,                /* MOV ES:W/BX/,AX    */
                      05DH,                          /* POP BP             */
                      0C2H,004H,000H),               /* RET 4              */
                     INIT$INT$VECTOR$CODE$PTR WORD DATA (.INIT$INT$VECTOR$CODE);

281   2           CALL INIT$INT$VECTOR$CODE$PTR(@VECTOR.SEG);    /* SEGMENT PORTION */
                                          ; STATEMENT # 281
          0874  C45E06          LES     BX,[BP].INTVECTORPTR
          0877  268D4702        LEA     AX,ES:VECTOR[BX+2H]
          087B  06              PUSH    ES      ; 1
          087C  50              PUSH    AX      ; 2
          087D  2EFF160600      CALL    CS:INITINTVECTORCODEPTR
282   2           VECTOR.OFF = INT$ROUTINE$OFFSET - CORRECTION;   /* OFFSET PORTION */
                                          ; STATEMENT # 282
          0882  8B4604          MOV     AX,[BP].INTROUTINEOFFSET
          0885  83E819          SUB     AX,19H
          0888  C45E06          LES     BX,[BP].INTVECTORPTR
          088B  268907          MOV     ES:VECTOR[BX],AX
283   2            END;
                                          ; STATEMENT # 283
          088E  5D              POP     BP
          088F  C20600          RET     6H
                 INITINTVECTOR      ENDP


     /* ************************************************************************
        ************************************************************************

                              COMMAND MODULE
                              ==============

        ABSTRACT
        ========
        THIS MODULE CONTAINS ALL THE COMMANDS IMPLEMENTED AS INDIVIDUAL PROCEDURES
        AND CALLED FROM THE OUTER BLOCK OF THE COMMAND DISPATCH LOOP.

        MODULE ORGANIZATION
        ===================
        THIS MODULE CONTAINS THE FOLLOWING SECTIONS:

           1. COMMANDS SECTION
                SIO$GO                  GO
                SIO$SINGLE$STEP         SINGLE STEP
                SIO$EXAM$MEM            SUBSTITUTE MEMORY
                SIO$EXAM$REG            EXAMINE REGISTER
                SIO$MOVE                MOVE
                SIO$DISPLAY             DISPLAY BYTES
```

```
                    SIO$INPUT           INPUT PORT
                    SIO$OUTPUT          OUTPUT PORT
                    SIO$WRITE           WRITE DATA RECORDS
                    SIO$READ            READ DATA RECORDS
                2. COMMAND DISPATCH (OUTER BLOCK, MAIN PROGRAM LOOP)
                    NEXT$COMMAND        DISPATCH
                    ERROR               ERROR ROUTINE
            */

               /************************
               *   COMMAND SECTION    *
               ************************/


284   1       SIO$GO:
                                                    ; STATEMENT # 284
               /* IMPLEMENTS THE 'GO' COMMAND. THE USER MAY SPECIFY A NEW
                  IP:PC AND AN OPTIONAL BREAKPOINT. */
                    SIOGO       PROC NEAR
        0892  55              PUSH    BP
        0893  8BEC            MOV     BP,SP
                PROCEDURE;
285   2            CALL SIO$UPDATE$IP;
                                                    ; STATEMENT # 285
        0895  E88DFD          CALL    SIOUPDATEIP
286   2            IF CHAR=',' THEN                 /* BREAKPOINT */
                                                    ; STATEMENT # 286
        0898  803E66002C      CMP     CHAR,2CH
        089D  753C            JNZ     @34
287   2                  DO;
288   3                  CALL SIO$GET$CHAR;
                                                    ; STATEMENT # 288
        089F  E86DFA          CALL    SIOGETCHAR
289   3                  CALL SIO$GET$ADDR(@BRK1,CSIP.SEG);
                                                    ; STATEMENT # 289
        08A2  8D062200        LEA     AX,BRK1
        08A6  1E              PUSH    DS      ; 1
        08A7  50              PUSH    AX      ; 2
        08A8  FF362800        PUSH    CSIP+2H ; 3
        08AC  E835FD          CALL    SIOGETADDR
290   3                IF CHAR<>ASCR THEN  GOTO ERROR;
                                                    ; STATEMENT # 290
        08AF  803E66000D      CMP     CHAR,0DH
        08B4  7403            JZ      $+5H
        08B6  E94DF9          JMP     ERROR
292   3                BRK1$SAVE = MEMORY$BRK1;
                                                    ; STATEMENT # 292
        08B9  C41E2200        LES     BX,MEMORYBRK1PTR
        08BD  268A07          MOV     AL,ES:MEMORYBRK1[BX]
        08C0  A26500          MOV     BRK1SAVE,AL
293   3                MEMORY$BRK1 = BREAK$INST;
                                                    ; STATEMENT # 293
        08C3  26C607CC        MOV     ES:MEMORYBRK1[BX],0CCH
294   3                IF MEMORY$BRK1<>BREAK$INST THEN GOTO ERROR;
                                                    ; STATEMENT # 294
        08C7  C41E2200        LES     BX,MEMORYBRK1PTR
        08CB  26803FCC        CMP     ES:MEMORYBRK1[BX],0CCH
```

```
          08CF  7403            JZ      $+5H
          08D1  E932F9          JMP     ERROR
296   3               BRK1$FLAG = TRUE;
                                          ; STATEMENT # 296
          08D4  C6066400FF      MOV     BRK1FLAG,0FFH
297   3               END;
                                          ; STATEMENT # 297
          08D9  EB0A            JMP     @37
                    @34:
                ELSE                        /* NO BREAKPOINT */
298   2               IF CHAR<>ASCR THEN  GOTO ERROR;
                                          ; STATEMENT # 298
          08DB  803E66000D      CMP     CHAR,0DH
          08E0  7403            JZ      $+5H
          08E2  E921F9          JMP     ERROR
                                          ; STATEMENT # 299
                    @37:
                CALL SIO$CRLF;
                                          ; STATEMENT # 300
          08E5  E8BFFB          CALL    SIOCRLF
301   2           IP = CSIP.OFF;
                                          ; STATEMENT # 301
          08E8  A12600          MOV     AX,CSIP
          08EB  A34800          MOV     REGSAV+18H,AX
302   2           CS = CSIP.SEG;
                                          ; STATEMENT # 302
          08EE  A12800          MOV     AX,CSIP+2H
          08F1  A34000          MOV     REGSAV+10H,AX
303   2           FL = FL AND (NOT STEP$TRAP);      /* CLEAR IF SET */
                                          ; STATEMENT # 303
          08F4  81264A00FFFE    AND     REGSAV+1AH,0FEFFH
304   2           CALL RESTORE$EXECUTE;
                                          ; STATEMENT # 304
          08FA  E869FE          CALL    RESTOREEXECUTE
305   2            END;
                                          ; STATEMENT # 305
          08FD  5D              POP     BP
          08FE  C3              RET
                    SIOGO       ENDP

306   1       SIO$SINGLE$STEP:
                                          ; STATEMENT # 306
                  /* IMPLEMENTS THE SINGLE STEP COMMAND. DISPLAYS IP AND THE
                     CURRENT INSTRUCTION BTYE. OPENS CS:IP FOR INPUT. DEPRESSING
                     COMMA CAUSES THE MONITOR TO SINGLE STEP THE INSTRUCTION, AND
                     PERIOD TERMINATES THE COMMAND. */
                        SIOSINGLESTEP       PROC NEAR
          08FF  55              PUSH    BP
          0900  8BEC            MOV     BP,SP
                 PROCEDURE;
307   2           CALL SIO$UPDATE$IP;
                                          ; STATEMENT # 307
          0902  E820FD          CALL    SIOUPDATEIP
308   2           IF CHAR<>',' THEN GOTO ERROR;
                                          ; STATEMENT # 308
          0905  803E66002C      CMP     CHAR,2CH
          090A  7403            JZ      $+5H
```

```
          090C  E9F7F8          JMP     ERROR
                    IP = CSIP.OFF;
                                                ; STATEMENT # 310
          090F  A12600          MOV     AX,CSIP
          0912  A34800          MOV     REGSAV+18H,AX
311   2             CS = CSIP.SEG;
                                                ; STATEMENT # 311
          0915  A12800          MOV     AX,CSIP+2H
          0918  A34000          MOV     REGSAV+10H,AX
312   2             FL = FL OR STEP$TRAP;
                                                ; STATEMENT # 312
          091B  810E4A000001    OR      REGSAV+1AH,100H
313   2             CALL RESTORE$EXECUTE;
                                                ; STATEMENT # 313
          0921  E842FE          CALL    RESTOREEXECUTE
314   2           END;
                                                ; STATEMENT # 314
          0924  5D              POP     BP
          0925  C3              RET
                          SIOSINGLESTEP     ENDP

315   1       SIO$EXAM$MEM:
                                                ; STATEMENT # 315
                  /* IMPLEMENTS THE EXAMINE MEMORY COMMAND. */
                          SIOEXAMMEM      PROC NEAR
          0926  55              PUSH    BP
          0927  8BEC            MOV     BP,SP
                  PROCEDURE;
316   2             DECLARE W WORD;
317   2             CALL SIO$TEST$WORD$MODE;
                                                ; STATEMENT # 317
          0929  E88CFB          CALL    SIOTESTWORDMODE
318   2             CALL SIO$GET$ADDR(@ARG1,CS);
                                                ; STATEMENT # 318
          092C  8D061A00        LEA     AX,ARG1
          0930  1E              PUSH    DS      ; 1
          0931  50              PUSH    AX      ; 2
          0932  FF364000        PUSH    REGSAV+10H; 3
          0936  E8ABFC          CALL    SIOGETADDR
319   2             IF CHAR<>',' THEN  GOTO ERROR;
                                                ; STATEMENT # 319
          0939  803E66002C      CMP     CHAR,2CH
          093E  7403            JZ      $+5H
          0940  E9C3F8          JMP     ERROR
321   2             DO WHILE TRUE;
                                                ; STATEMENT # 321
                      @148:
322   3                 CALL SIO$OUT$BLANK;
                                                ; STATEMENT # 322
          0943  E831FA          CALL    SIOOUTBLANK
323   3                 IF WORD$MODE THEN
                                                ; STATEMENT # 323
          0946  A06900          MOV     AL,WORDMODE
          0949  D0D8            RCR     AL,1
          094B  730C            JNB     @41
324   3                   CALL SIO$OUT$WORD(MEMORY$WORD$ARG1);
                                                ; STATEMENT # 324
```

```
          094D  C41E1A00        LES     BX,MEMORYARG1PTR
          0951  26FF37          PUSH    ES:MEMORYWORDARG1[BX]
          0954  E809FA          CALL    SIOOUTWORD
          0957  EB0A            JMP     @42
                        @41:
                        ELSE
325   3                   CALL SIO$OUT$BYTE(MEMORY$ARG1);
                                                  ; STATEMENT # 325
          0959  C41E1A00        LES     BX,MEMORYARG1PTR
          095D  26FF37          PUSH    ES:MEMORYARG1[BX]
          0960  E8CDF9          CALL    SIOOUTBYTE
                        @42:
326   3                   CALL SIO$OUT$CHAR('-');
                                                  ; STATEMENT # 326
          0963  B02D            MOV     AL,2DH
          0965  50              PUSH    AX      ; 1
          0966  E886F9          CALL    SIOOUTCHAR
327   3                   CALL SIO$OUT$BLANK;
                                                  ; STATEMENT # 327
          0969  E80BFA          CALL    SIOOUTBLANK
328   3                   CALL SIO$GET$CHAR;
                                                  ; STATEMENT # 328
          096C  E8A0F9          CALL    SIOGETCHAR
329   3                   IF CHAR=ASCR THEN RETURN;
                                                  ; STATEMENT # 329
          096F  803E66000D      CMP     CHAR,0DH
          0974  7502            JNZ     @43
                                                  ; STATEMENT # 330
          0976  5D              POP     BP
          0977  C3              RET
                        @43:
331   3                   IF CHAR<>',' THEN
                                                  ; STATEMENT # 331
          0978  803E66002C      CMP     CHAR,2CH
          097D  745C            JZ      @44
332   3                         DO;
333   4                         W = SIO$GET$WORD;
                                                  ; STATEMENT # 333
          097F  E86EFB          CALL    SIOGETWORD
          0982  A35200          MOV     W,AX
334   4                         IF (CHAR <> ',') AND (CHAR <> ASCR) THEN GOTO ERROR;
                                                  ; STATEMENT # 334
          0985  803E66002C      CMP     CHAR,2CH
          098A  B0FF            MOV     AL,0FFH
          098C  7501            JNZ     $+3H
          098E  40              INC     AX
          098F  50              PUSH    AX      ; 1
          0990  803E66000D      CMP     CHAR,0DH
          0995  B0FF            MOV     AL,0FFH
          0997  7501            JNZ     $+3H
          0999  40              INC     AX
          099A  59              POP     CX      ; 1
          099B  22C1            AND     AL,CL
          099D  D0D8            RCR     AL,1
          099F  7303            JNB     $+5H
          09A1  E962F8          JMP     ERROR
336   4                         IF WORD$MODE THEN
```

```
                                                    ; STATEMENT # 336
          09A4  A06900          MOV     AL,WORDMODE
          09A7  D0D8            RCR     AL,1
          09A9  7317            JNB     @46
337   4                     DO;
338   5                     MEMORY$WORD$ARG1 = W;
                                                    ; STATEMENT # 338
          09AB  A15200          MOV     AX,W
          09AE  C41E1A00        LES     BX,MEMORYARG1PTR
          09B2  268907          MOV     ES:MEMORYWORDARG1[BX],AX
339   5                     IF MEMORY$WORD$ARG1<>W THEN GOTO ERROR;
                                                    ; STATEMENT # 339
          09B5  C41E1A00        LES     BX,MEMORYARG1PTR
          09B9  268B07          MOV     AX,ES:MEMORYWORDARG1[BX]
          09BC  3B065200        CMP     AX,W
          09C0  EB14            JMP     @11
341   5                     END;
                                                    ; STATEMENT # 341
                        @46:
                          ELSE
342   4                     DO;
343   5                     MEMORY$ARG1 = LOW(W);
                                                    ; STATEMENT # 343
          09C2  A15200          MOV     AX,W
          09C5  C41E1A00        LES     BX,MEMORYARG1PTR
          09C9  268807          MOV     ES:MEMORYARG1[BX],AL
344   5                     IF MEMORY$ARG1<>LOW(W) THEN GOTO ERROR;
                                                    ; STATEMENT # 344
          09CC  A15200          MOV     AX,W
          09CF  C41E1A00        LES     BX,MEMORYARG1PTR
          09D3  263807          CMP     ES:MEMORYARG1[BX],AL
                        @11:
          09D6  7403            JZ      $+5H
          09D8  E92BF8          JMP     ERROR
                                                    ; STATEMENT # 345
346   5                     END;
347   4                   END;
                        @44:
348   3                 IF CHAR=ASCR THEN RETURN;
                                                    ; STATEMENT # 348
          09DB  803E66000D      CMP     CHAR,0DH
          09E0  7502            JNZ     @50
                                                    ; STATEMENT # 349
          09E2  5D              POP     BP
          09E3  C3              RET
                        @50:
350   3                 IF WORD$MODE THEN
                                                    ; STATEMENT # 350
          09E4  A06900          MOV     AL,WORDMODE
          09E7  D0D8            RCR     AL,1
          09E9  7307            JNB     @51
351   3                   ARG1.OFF = ARG1.OFF + 2;
                                                    ; STATEMENT # 351
          09EB  83061A0002      ADD     ARG1,2H
          09F0  EB05            JMP     @52
                        @51:
                        ELSE
```

```
352   3                 ARG1.OFF = ARG1.OFF + 1;
                                                  ; STATEMENT # 352
          09F2  83061A0001      ADD     ARG1,1H
                      @52:
353   3               CALL SIO$CRLF;
                                                  ; STATEMENT # 353
          09F7  E8ADFA          CALL    SIOCRLF
354   3               CALL SIO$OUT$WORD(ARG1.OFF);
                                                  ; STATEMENT # 354
          09FA  FF361A00        PUSH    ARG1    ; 1
          09FE  E85FF9          CALL    SIOOUTWORD
355   3           END;
                                                  ; STATEMENT # 355
          0A01  E93FFF          JMP     @148
                      @149:
356   2         END;
                                                  ; STATEMENT # 356
          0A04  5D              POP     BP
          0A05  C3              RET
                    SIOEXAMMEM      ENDP

357   1       SIO$EXAM$REG:
                                                  ; STATEMENT # 357
                /* IMPLEMENTS THE EXAMINE REGISTER COMMAND. SCANS FOR A VALID
                   REGISTER NAME AND DISPLAYS THE VALUE OF THAT REGISTER WHICH IS
                   OPTIONALLY OPENED FOR INPUT. COMMA INCREMENTS TO NEXT REGISTER
                   UNLESS IT IS 'FL' WHICH TERMINATES AS DOES CR. */
                    SIOEXAMREG      PROC NEAR
          0A06  55              PUSH    BP
          0A07  8BEC            MOV     BP,SP
           PROCEDURE;
358   2          DECLARE (T,I) BYTE, SAVE WORD;
359   2          CALL SIO$SCAN$BLANK;
                                                  ; STATEMENT # 359
          0A09  E8D2FA          CALL    SIOSCANBLANK
360   2          IF CHAR=ASCR THEN
                                                  ; STATEMENT # 360
          0A0C  803E66000D      CMP     CHAR,0DH
          0A11  7559            JNZ     @53
361   2                DO;
362   3                CALL SIO$CRLF;
                                                  ; STATEMENT # 362
          0A13  E891FA          CALL    SIOCRLF
363   3                DO I=0 TO 13;
                                                  ; STATEMENT # 363
          0A16  C606700000      MOV     I,0H
                      @150:
          0A1B  803E70000D      CMP     I,0DH
          0A20  7748            JA      @151
364   4                  CALL SIO$OUT$BLANK;
                                                  ; STATEMENT # 364
          0A22  E852F9          CALL    SIOOUTBLANK
365   4                  CALL SIO$OUT$CHAR(REG(I*2));
                                                  ; STATEMENT # 365
          0A25  A07000          MOV     AL,I
          0A28  B102            MOV     CL,2H
          0A2A  F6E1            MUL     CL
```

```
          0A2C  89C3            MOV     BX,AX
          0A2E  2EFF7758        PUSH    CS:REG[BX]
          0A32  E8BAF8          CALL    SIOOUTCHAR
366   4                 CALL SIO$OUT$CHAR(REG(I*2+1));
                                                  ; STATEMENT # 366
          0A35  A07000          MOV     AL,I
          0A38  B102            MOV     CL,2H
          0A3A  F6E1            MUL     CL
          0A3C  89C3            MOV     BX,AX
          0A3E  2EFF7759        PUSH    CS:REG[BX+1H]
          0A42  E8AAF8          CALL    SIOOUTCHAR
367   4                 CALL SIO$OUT$CHAR('=');
                                                  ; STATEMENT # 367
          0A45  B03D            MOV     AL,3DH
          0A47  50              PUSH    AX      ; 1
          0A48  E8A4F8          CALL    SIOOUTCHAR
368   4                 CALL SIO$OUT$WORD(REG$SAV(I));
                                                  ; STATEMENT # 368
          0A4B  8A1E7000        MOV     BL,I
          0A4F  B700            MOV     BH,0H
          0A51  D1E3            SHL     BX,1
          0A53  FF7730          PUSH    REGSAV[BX]; 1
          0A56  E807F9          CALL    SIOOUTWORD
369   4                 IF I=6 THEN CALL SIO$CRLF;
                                                  ; STATEMENT # 369
          0A59  803E700006      CMP     I,6H
          0A5E  7503            JNZ     @54
                                                  ; STATEMENT # 370
          0A60  E844FA          CALL    SIOCRLF
                          @54:
371   4                 END;
                                                  ; STATEMENT # 371
          0A63  8006700001      ADD     I,1H
          0A68  75B1            JNZ     @150
                          @151:
372   3                 RETURN;
                                                  ; STATEMENT # 372
          0A6A  5D              POP     BP
          0A6B  C3              RET
373   3                 END;
                          @53:
374   2           IF NOT(SIO$VALID$REG$FIRST) THEN  GOTO ERROR;
                                                  ; STATEMENT # 374
          0A6C  E8C2F9          CALL    SIOVALIDREGFIRST
          0A6F  F6D0            NOT     AL
          0A71  D0D8            RCR     AL,1
          0A73  7303            JNB     $+5H
          0A75  E98EF7          JMP     ERROR
376   2               T = CHAR;
                                                  ; STATEMENT # 376
          0A78  A06600          MOV     AL,CHAR
          0A7B  A26F00          MOV     T,AL
377   2           CALL SIO$GET$CHAR;
                                                  ; STATEMENT # 377
          0A7E  E88EF8          CALL    SIOGETCHAR
378   2           IF NOT(SIO$VALID$REG(T,CHAR)) THEN  GOTO ERROR;
                                                  ; STATEMENT # 378
```

```
                0A81  FF366F00          PUSH    T       ; 1
                0A85  FF366600          PUSH    CHAR    ; 2
                0A89  E8D4F9            CALL    SIOVALIDREG
                0A8C  F6D0              NOT     AL
                0A8E  D0D8              RCR     AL,1
                0A90  7303              JNB     $+5H
                0A92  E971F7            JMP     ERROR
380   2             I = REG$INDEX;
                                                ; STATEMENT # 380
                0A95  A12E00            MOV     AX,REGINDEX
                0A98  A27000            MOV     I,AL
381   2             DO WHILE TRUE;
                                                ; STATEMENT # 381
                       @152:
382   3               CALL SIO$OUT$CHAR('=');
                                                ; STATEMENT # 382
                0A9B  B03D              MOV     AL,3DH
                0A9D  50                PUSH    AX      ; 1
                0A9E  E84EF8            CALL    SIOOUTCHAR
383   3               CALL SIO$OUT$WORD(REG$SAV(I));
                                                ; STATEMENT # 383
                0AA1  8A1E7000          MOV     BL,I
                0AA5  B700              MOV     BH,0H
                0AA7  D1E3              SHL     BX,1
                0AA9  FF7730            PUSH    REGSAV[BX]; 1
                0AAC  E8B1F8            CALL    SIOOUTWORD
384   3               CALL SIO$OUT$CHAR('-');
                                                ; STATEMENT # 384
                0AAF  B02D              MOV     AL,2DH
                0AB1  50                PUSH    AX      ; 1
                0AB2  E83AF8            CALL    SIOOUTCHAR
385   3               CALL SIO$OUT$BLANK;
                                                ; STATEMENT # 385
                0AB5  E8BFF8            CALL    SIOOUTBLANK
386   3               CALL SIO$GET$CHAR;
                                                ; STATEMENT # 386
                0AB8  E854F8            CALL    SIOGETCHAR
387   3               IF CHAR<>',' AND CHAR<>ASCR THEN
                                                ; STATEMENT # 387
                0ABB  803E66002C        CMP     CHAR,2CH
                0AC0  B0FF              MOV     AL,0FFH
                0AC2  7501              JNZ     $+3H
                0AC4  40                INC     AX
                0AC5  50                PUSH    AX      ; 1
                0AC6  803E66000D        CMP     CHAR,0DH
                0ACB  B0FF              MOV     AL,0FFH
                0ACD  7501              JNZ     $+3H
                0ACF  40                INC     AX
                0AD0  59                POP     CX      ; 1
                0AD1  22C1              AND     AL,CL
                0AD3  D0D8              RCR     AL,1
                0AD5  7333              JNB     @57
388   3                 DO;
389   4                 SAVE = SIO$GET$WORD;
                                                ; STATEMENT # 389
                0AD7  E816FA            CALL    SIOGETWORD
                0ADA  A35400            MOV     SAVE,AX
```

```
390   4                 IF (CHAR <> ',' ) AND (CHAR <> ASCR) THEN GOTO ERROR;
                                                        ; STATEMENT # 390
      0ADD  803E66002C        CMP     CHAR,2CH
      0AE2  B0FF              MOV     AL,0FFH
      0AE4  7501              JNZ     $+3H
      0AE6  40                INC     AX
      0AE7  50                PUSH    AX        ; 1
      0AE8  803E66000D        CMP     CHAR,0DH
      0AED  B0FF              MOV     AL,0FFH
      0AEF  7501              JNZ     $+3H
      0AF1  40                INC     AX
      0AF2  59                POP     CX        ; 1
      0AF3  22C1              AND     AL,CL
      0AF5  D0D8              RCR     AL,1
      0AF7  7303              JNB     $+5H
      0AF9  E90AF7            JMP     ERROR
392   4                   REG$SAV(I) = SAVE;
                                                        ; STATEMENT # 392
      0AFC  8A1E7000          MOV     BL,I
      0B00  B700              MOV     BH,0H
      0B02  D1E3              SHL     BX,1
      0B04  A15400            MOV     AX,SAVE
      0B07  894730            MOV     REGSAV[BX],AX
393   4                   END;
                        @57:
394   3                 IF CHAR=ASCR OR I=13 THEN RETURN;
                                                        ; STATEMENT # 394
      0B0A  803E66000D        CMP     CHAR,0DH
      0B0F  B0FF              MOV     AL,0FFH
      0B11  7401              JZ      $+3H
      0B13  40                INC     AX
      0B14  50                PUSH    AX        ; 1
      0B15  803E70000D        CMP     I,0DH
      0B1A  B0FF              MOV     AL,0FFH
      0B1C  7401              JZ      $+3H
      0B1E  40                INC     AX
      0B1F  59                POP     CX        ; 1
      0B20  0AC1              OR      AL,CL
      0B22  D0D8              RCR     AL,1
      0B24  7302              JNB     @59
                                                        ; STATEMENT # 395
      0B26  5D                POP     BP
      0B27  C3                RET
                        @59:
396   3                 I = I + 1;
                                                        ; STATEMENT # 396
      0B28  8006700001        ADD     I,1H
397   3                 CALL SIO$CRLF;
                                                        ; STATEMENT # 397
      0B2D  E877F9            CALL    SIOCRLF
398   3                 CALL SIO$OUT$CHAR(REG(I*2));
                                                        ; STATEMENT # 398
      0B30  A07000            MOV     AL,I
      0B33  B102              MOV     CL,2H
      0B35  F6E1              MUL     CL
      0B37  89C3              MOV     BX,AX
      0B39  2EFF7758          PUSH    CS:REG[BX]
```

```
                0B3D  E8AFF7          CALL    SIOOUTCHAR
399   3                   CALL SIO$OUT$CHAR(REG(I*2+1));
                                              ; STATEMENT # 399
                0B40  A07000          MOV     AL,I
                0B43  B102            MOV     CL,2H
                0B45  F6E1            MUL     CL
                0B47  89C3            MOV     BX,AX
                0B49  2EFF7759        PUSH    CS:REG[BX+1H]
                0B4D  E89FF7          CALL    SIOOUTCHAR
400   3                   END;
                                              ; STATEMENT # 400
                0B50  E948FF          JMP     @152
                          @153:
401   2                  END;
                                              ; STATEMENT # 401
                0B53  5D              POP     BP
                0B54  C3              RET
                          SIOEXAMREG  ENDP

402   1              SIO$MOVE:
                                              ; STATEMENT # 402
                       /* IMPLEMENTS THE MOVE COMMAND. ACCEPTS 3 ARGUMENTS AMD MOVES THE
                          BLOCK OF MEMORY SPECIFIED BY ARG1-ARG2 TO ARG3. ARG2<ARG1 OR THERE
                          IS A DIFFERENCE WHEN THE BYTE IS READ BACK, THEN ERROR. */
                           SIOMOVE      PROC NEAR
                0B55  55              PUSH    BP
                0B56  8BEC            MOV     BP,SP
                       PROCEDURE;
403   2                   CALL SIO$SCAN$BLANK;
                                              ; STATEMENT # 403
                0B58  E883F9          CALL    SIOSCANBLANK
404   2                   CALL SIO$GET$ADDR(@ARG1,CS);          /* FIRST ARGUMENT */
                                              ; STATEMENT # 404
                0B5B  8D061A00        LEA     AX,ARG1
                0B5F  1E              PUSH    DS      ; 1
                0B60  50              PUSH    AX      ; 2
                0B61  FF364000        PUSH    REGSAV+10H; 3
                0B65  E87CFA          CALL    SIOGETADDR
405   2                   IF CHAR<>',' THEN GOTO ERROR;
                                              ; STATEMENT # 405
                0B68  803E66002C      CMP     CHAR,2CH
                0B6D  7403            JZ      $+5H
                0B6F  E994F6          JMP     ERROR
407   2                   CALL SIO$GET$CHAR;
                                              ; STATEMENT # 407
                0B72  E89AF7          CALL    SIOGETCHAR
408   2                   END$OFF = SIO$GET$WORD;               /* SECOND ARGUMENT */
                                              ; STATEMENT # 408
                0B75  E878F9          CALL    SIOGETWORD
                0B78  A31800          MOV     ENDOFF,AX
409   2                   IF END$OFF<ARG1.OFF THEN  GOTO ERROR;
                                              ; STATEMENT # 409
                0B7B  A11800          MOV     AX,ENDOFF
                0B7E  3B061A00        CMP     AX,ARG1
                0B82  7303            JNB     $+5H
                0B84  E97FF6          JMP     ERROR
411   2                   IF CHAR<>',' THEN  GOTO ERROR;
```

```
                                                    ; STATEMENT # 411
        0B87  803E66002C        CMP     CHAR,2CH
        0B8C  7403              JZ      $+5H
        0B8E  E975F6            JMP     ERROR
413  2              CALL SIO$GET$CHAR;
                                                    ; STATEMENT # 413
        0B91  E87BF7            CALL    SIOGETCHAR
414  2              CALL SIO$GET$ADDR(@ARG3,CS);     /* THIRD ARGUMENT */
                                                    ; STATEMENT # 414
        0B94  8D061E00          LEA     AX,ARG3
        0B98  1E                PUSH    DS      ; 1
        0B99  50                PUSH    AX      ; 2
        0B9A  FF364000          PUSH    REGSAV+10H; 3
        0B9E  E843FA            CALL    SIOGETADDR
415  2              IF CHAR<>ASCR THEN GOTO ERROR;
                                                    ; STATEMENT # 415
        0BA1  803E66000D        CMP     CHAR,0DH
        0BA6  7403              JZ      $+5H
        0BA8  E95BF6            JMP     ERROR
417  2              CALL SIO$CRLF;
                                                    ; STATEMENT # 417
        0BAB  E8F9F8            CALL    SIOCRLF
418  2          LOOP:
                                                    ; STATEMENT # 418
                       LOOP:
                   MEMORY$ARG3 = MEMORY$ARG1;
        0BAE  C41E1A00          LES     BX,MEMORYARG1PTR
        0BB2  268A07            MOV     AL,ES:MEMORYARG1[BX]
        0BB5  C41E1E00          LES     BX,MEMORYARG3PTR
        0BB9  268807            MOV     ES:MEMORYARG3[BX],AL
419  2              IF MEMORY$ARG3<>MEMORY$ARG1 THEN GOTO ERROR;
                                                    ; STATEMENT # 419
        0BBC  C41E1E00          LES     BX,MEMORYARG3PTR
        0BC0  268A07            MOV     AL,ES:MEMORYARG3[BX]
        0BC3  C41E1A00          LES     BX,MEMORYARG1PTR
        0BC7  263A07            CMP     AL,ES:MEMORYARG1[BX]
        0BCA  7403              JZ      $+5H
        0BCC  E937F6            JMP     ERROR
421  2              IF ARG1.OFF = END$OFF THEN RETURN;
                                                    ; STATEMENT # 421
        0BCF  A11A00            MOV     AX,ARG1
        0BD2  3B061800          CMP     AX,ENDOFF
        0BD6  7502              JNZ     @65
                                                    ; STATEMENT # 422
        0BD8  5D                POP     BP
        0BD9  C3                RET
                       @65:
423  2              ARG1.OFF = ARG1.OFF + 1;
                                                    ; STATEMENT # 423
        0BDA  83061A0001        ADD     ARG1,1H
424  2              ARG3.OFF = ARG3.OFF + 1;
                                                    ; STATEMENT # 424
        0BDF  83061E0001        ADD     ARG3,1H
425  2              GOTO LOOP;
                                                    ; STATEMENT # 425
        0BE4  EBC8              JMP     LOOP
426  2            END;
```

```
                        SIOMOVE      ENDP

427   1       SIO$DISPLAY:
                                                    ; STATEMENT # 427
                  /* IMPLEMENTS THE DISPLAY BYTE COMMAND. IF CALLED WITH 1 PARM THEN
                       OUTPUTS A SINGLE BYTE. IF CALLED WITH 2 PARMS THEN OUTPUTS THE RANGE
                       BETWEEN THE TWO ADDRESSES. IF OFFSET<BEGIN THEN OUTPUTS ONLY A SINGLE
                       BYTE. */
                          SIODISPLAY      PROC NEAR
          0BE6  55                PUSH    BP
          0BE7  8BEC              MOV     BP,SP
                   PROCEDURE;
428   2             DECLARE T BYTE;
429   2             CALL SIO$TEST$WORD$MODE;
                                                    ; STATEMENT # 429
          0BE9  E8CCF8            CALL    SIOTESTWORDMODE
430   2             CALL SIO$GET$ADDR(@ARG1,CS);
                                                    ; STATEMENT # 430
          0BEC  8D061A00          LEA     AX,ARG1
          0BF0  1E                PUSH    DS      ; 1
          0BF1  50                PUSH    AX      ; 2
          0BF2  FF364000          PUSH    REGSAV+10H; 3
          0BF6  E8EBF9            CALL    SIOGETADDR
431   2             IF CHAR=ASCR THEN
                                                    ; STATEMENT # 431
          0BF9  803E66000D        CMP     CHAR,0DH
          0BFE  7508              JNZ     @66
432   2                   END$OFF = ARG1.OFF;
                                                    ; STATEMENT # 432
          0C00  A11A00            MOV     AX,ARG1
          0C03  A31800            MOV     ENDOFF,AX
          0C06  EB29              JMP     NEWLINE
                           @66:
                        ELSE
433   2                    DO;
434   3                    IF CHAR<>',' THEN GOTO ERROR;
                                                    ; STATEMENT # 434
          0C08  803E66002C        CMP     CHAR,2CH
          0C0D  7403              JZ      $+5H
          0C0F  E9F4F5            JMP     ERROR
436   3                    CALL SIO$GET$CHAR;
                                                    ; STATEMENT # 436
          0C12  E8FAF6            CALL    SIOGETCHAR
437   3                    END$OFF = SIO$GET$WORD;
                                                    ; STATEMENT # 437
          0C15  E8D8F8            CALL    SIOGETWORD
          0C18  A31800            MOV     ENDOFF,AX
438   3                    IF END$OFF < ARG1.OFF THEN GOTO ERROR;
                                                    ; STATEMENT # 438
          0C1B  A11800            MOV     AX,ENDOFF
          0C1E  3B061A00          CMP     AX,ARG1
          0C22  7303              JNB     $+5H
          0C24  E9DFF5            JMP     ERROR
440   3                    IF CHAR<>ASCR THEN  GOTO ERROR;
                                                    ; STATEMENT # 440
          0C27  803E66000D        CMP     CHAR,0DH
          0C2C  7403              JZ      $+5H
```

```
          0C2E  E9D5F5          JMP     ERROR
442   3             END;
443   2       NEWLINE:
                                                ; STATEMENT # 443
                    NEWLINE:
                CALL SIO$CRLF;
          0C31  E873F8          CALL    SIOCRLF
444   2         CALL SIO$OUT$WORD(ARG1.OFF);
                                                ; STATEMENT # 444
          0C34  FF361A00        PUSH    ARG1    ; 1
          0C38  E825F7          CALL    SIOOUTWORD
445   2     LOOP: CALL SIO$OUT$BLANK;
                                                ; STATEMENT # 445
                  LOOP:
          0C3B  E839F7          CALL    SIOOUTBLANK
446   2         IF WORD$MODE THEN
                                                ; STATEMENT # 446
          0C3E  A06900          MOV     AL,WORDMODE
          0C41  D0D8            RCR     AL,1
          0C43  731C            JNB     @71
447   2             DO;
448   3             CALL SIO$OUT$WORD(MEMORY$WORD$ARG1);
                                                ; STATEMENT # 448
          0C45  C41E1A00        LES     BX,MEMORYARG1PTR
          0C49  26FF37          PUSH    ES:MEMORYWORDARG1[BX]
          0C4C  E811F7          CALL    SIOOUTWORD
449   3             IF ARG1.OFF = END$OFF THEN RETURN;
                                                ; STATEMENT # 449
          0C4F  A11A00          MOV     AX,ARG1
          0C52  3B061800        CMP     AX,ENDOFF
          0C56  7502            JNZ     @72
                                                ; STATEMENT # 450
          0C58  5D              POP     BP
          0C59  C3              RET
                  @72:
451   3             ARG1.OFF = ARG1.OFF + 1;
                                                ; STATEMENT # 451
          0C5A  83061A0001      ADD     ARG1,1H
452   3             END;
                                                ; STATEMENT # 452
          0C5F  EB0A            JMP     @73
                  @71:
              ELSE
453   2             CALL SIO$OUT$BYTE(MEMORY$ARG1);
                                                ; STATEMENT # 453
          0C61  C41E1A00        LES     BX,MEMORYARG1PTR
          0C65  26FF37          PUSH    ES:MEMORYARG1[BX]
          0C68  E8C5F6          CALL    SIOOUTBYTE
                  @73:
454   2         IF ARG1.OFF>=END$OFF THEN RETURN;
                                                ; STATEMENT # 454
          0C6B  A11A00          MOV     AX,ARG1
          0C6E  3B061800        CMP     AX,ENDOFF
          0C72  7202            JB      @74
                                                ; STATEMENT # 455
          0C74  5D              POP     BP
          0C75  C3              RET
```

```
                    @74:
456   2               ARG1.OFF = ARG1.OFF + 1;
                                                    ; STATEMENT # 456
          0C76  83061A0001      ADD     ARG1,1H
457   2               T = ARG1.OFF AND 000FH;
                                                    ; STATEMENT # 457
          0C7B  A11A00          MOV     AX,ARG1
          0C7E  250F00          AND     AX,0FH
          0C81  A27100          MOV     T,AL
458   2               IF T=0 OR (WORD$MODE AND T=1) THEN GOTO NEWLINE;
                                                    ; STATEMENT # 458
          0C84  823E710000      CMP     T,0H
          0C89  B0FF            MOV     AL,0FFH
          0C8B  7401            JZ      $+3H
          0C8D  40              INC     AX
          0C8E  50              PUSH    AX        ; 1
          0C8F  803E710001      CMP     T,1H
          0C94  B0FF            MOV     AL,0FFH
          0C96  7401            JZ      $+3H
          0C98  40              INC     AX
          0C99  22066900        AND     AL,WORDMODE
          0C9D  59              POP     CX        ; 1
          0C9E  0AC1            OR      AL,CL
          0CA0  D0D8            RCR     AL,1
          0CA2  728D            JB      NEWLINE
460   2               GOTO LOOP;
                                                    ; STATEMENT # 460
          0CA4  EB95            JMP     LOOP
461   2               END;
                        SIODISPLAY      ENDP

462   1         SIO$INPUT:
                                                    ; STATEMENT # 462
                   /* THIS ROUTINE IMPLEMENTS THE 'INPUT' COMMAND. USER SPECIFIES
                      A PORT AND THE DATUM OF THE PORT IS DISPLAYED. */
                        SIOINPUT        PROC NEAR
          0CA6  55              PUSH    BP
          0CA7  8BEC            MOV     BP,SP
                PROCEDURE;
463   2            DECLARE PORT WORD;
464   2            CALL SIO$TEST$WORD$MODE;
                                                    ; STATEMENT # 464
          0CA9  E80CF8          CALL    SIOTESTWORDMODE
465   2            PORT = SIO$GET$WORD;
                                                    ; STATEMENT # 465
          0CAC  E841F8          CALL    SIOGETWORD
          0CAF  A35600          MOV     PORT,AX
466   2         LOOP:
                                                    ; STATEMENT # 466
                        LOOP:
          0CB2  803E66002C      CMP     CHAR,2CH
          0CB7  7403            JZ      $+5H
          0CB9  E94AF5          JMP     ERROR
                   IF CHAR<>',' THEN GOTO ERROR;
468   2            CALL SIO$CRLF;
                                                    ; STATEMENT # 468
          0CBC  E8E8F7          CALL    SIOCRLF
```

```
469   2            IF WORD$MODE THEN
                                                ; STATEMENT # 469
          0CBF  A06900          MOV     AL,WORDMODE
          0CC2  D0D8            RCR     AL,1
          0CC4  730B            JNB     @77
470   2                CALL SIO$OUT$WORD(INWORD(PORT));
                                                ; STATEMENT # 470
          0CC6  8B165600        MOV     DX,PORT
          0CCA  ED              INW     DX
          0CCB  50              PUSH    AX      ; 1
          0CCC  E891F6          CALL    SIOOUTWORD
          0CCF  EB09            JMP     @78
                        @77:
                ELSE
471   2               CALL SIO$OUT$BYTE(INPUT(PORT));
                                                ; STATEMENT # 471
          0CD1  8B165600        MOV     DX,PORT
          0CD5  EC              IN      DX
          0CD6  50              PUSH    AX      ; 1
          0CD7  E856F6          CALL    SIOOUTBYTE
                        @78:
472   2            CALL SIO$GET$CHAR;
                                                ; STATEMENT # 472
          0CDA  E832F6          CALL    SIOGETCHAR
473   2            IF CHAR=ASCR THEN RETURN;
                                                ; STATEMENT # 473
          0CDD  803E66000D      CMP     CHAR,0DH
          0CE2  75CE            JNZ     LOOP
                                                ; STATEMENT # 474
          0CE4  5D              POP     BP
          0CE5  C3              RET
475   2            GOTO LOOP;
476   2          END;
                        SIOINPUT        ENDP

477   1       SIO$OUTPUT:
                                                ; STATEMENT # 477
                /* THIS ROUTINE IMPLEMENTS THE 'OUTPUT' COMMAND. THE USER SUPPLIED
                   DATUM IS OUTPUT TO THE SPECIFIED PORT. */
                        SIOOUTPUT       PROC NEAR
          0CE6  55              PUSH    BP
          0CE7  8BEC            MOV     BP,SP
                PROCEDURE;
478   2            DECLARE (DATUM,PORT) WORD;
479   2            CALL SIO$TEST$WORD$MODE;
                                                ; STATEMENT # 479
          0CE9  E8CCF7          CALL    SIOTESTWORDMODE
480   2            PORT = SIO$GET$WORD;
                                                ; STATEMENT # 480
          0CEC  E801F8          CALL    SIOGETWORD
          0CEF  A35A00          MOV     PORT,AX
481   2            IF CHAR<>',' THEN GOTO ERROR;
                                                ; STATEMENT # 481
          0CF2  803E66002C      CMP     CHAR,2CH
          0CF7  7403            JZ      $+5H
          0CF9  E90AF5          JMP     ERROR
483   2            CALL SIO$GET$CHAR;
```

```
                                                    ; STATEMENT # 483
      0CFC  E810F6          CALL    SIOGETCHAR
484   2       LOOP:
                                                    ; STATEMENT # 484
                    LOOP:
                DATUM = SIO$GET$WORD;
      0CFF  E8EEF7          CALL    SIOGETWORD
      0D02  A35800          MOV     DATUM,AX
485   2             IF CHAR=':' THEN GOTO ERROR;
                                                    ; STATEMENT # 485
      0D05  803E66003A      CMP     CHAR,3AH
      0D0A  7503            JNZ     $+5H
      0D0C  E9F7F4          JMP     ERROR
487   2             IF WORD$MODE THEN
                                                    ; STATEMENT # 487
      0D0F  A06900          MOV     AL,WORDMODE
      0D12  D0D8            RCR     AL,1
      0D14  730A            JNB     @82
488   2                   OUTWORD(PORT) = DATUM;
                                                    ; STATEMENT # 488
      0D16  8B165A00        MOV     DX,PORT
      0D1A  A15800          MOV     AX,DATUM
      0D1D  EF              OUTW    DX
      0D1E  EB08            JMP     @83
                    @82:
                ELSE
489   2                   OUTPUT(PORT) = LOW(DATUM);
                                                    ; STATEMENT # 489
      0D20  A15800          MOV     AX,DATUM
      0D23  8B165A00        MOV     DX,PORT
      0D27  EE              OUT     DX
                     @83:
490   2             IF CHAR=',' THEN
                                                    ; STATEMENT # 490
      0D28  803E66002C      CMP     CHAR,2CH
      0D2D  7516            JNZ     @84
491   2                   DO;
492   3                   CALL SIO$CRLF;
                                                    ; STATEMENT # 492
      0D2F  E875F7          CALL    SIOCRLF
493   3                   CALL SIO$OUT$CHAR('-');
                                                    ; STATEMENT # 493
      0D32  B02D            MOV     AL,2DH
      0D34  50              PUSH    AX      ; 1
      0D35  E8B7F5          CALL    SIOOUTCHAR
494   3                   CALL SIO$OUT$BLANK;
                                                    ; STATEMENT # 494
      0D38  E83CF6          CALL    SIOOUTBLANK
495   3                   CALL SIO$GET$CHAR;
                                                    ; STATEMENT # 495
      0D3B  E8D1F5          CALL    SIOGETCHAR
496   3                   IF CHAR <> ASCR THEN GOTO LOOP;
                                                    ; STATEMENT # 496
      0D3E  803E66000D      CMP     CHAR,0DH
      0D43  75BA            JNZ     LOOP
                                                    ; STATEMENT # 497
498   3                   END;
```

```
                        @84:
499   2           END;
                                                    ; STATEMENT # 499
            0D45  5D              POP     BP
            0D46  C3              RET
                        SIOOUTPUT       ENDP

500   1       SIO$WRITE:
                                                    ; STATEMENT # 500
                  /* IMPLEMENTS THE PAPER TAPE WRITE COMMAND. OUTPUTS LEADING NULLS,
                       EXTENDED ADDRESS RECORD (8086 ONLY), START ADDRESS RECORDS
                       (8086 ONLY), DATA RECORDS, EOF RECORD, AND TRAILING NULLS. */
                          SIOWRITE      PROC NEAR
            0D47  55              PUSH    BP
            0D48  8BEC            MOV     BP,SP
                   PROCEDURE;
501   2              DECLARE (START$REC,MODE$8086) BYTE, (LEN,INDEX) WORD;
502   2              CALL SIO$GET$CHAR;
                                                    ; STATEMENT # 502
            0D4A  E8C2F5          CALL    SIOGETCHAR
503   2              MODE$8086 = TRUE;
                                                    ; STATEMENT # 503
            0D4D  C6067300FF      MOV     MODE8086,0FFH
504   2              IF CHAR='X' THEN                    /* TEST FOR 8080 MODE */
                                                    ; STATEMENT # 504
            0D52  803E660058      CMP     CHAR,58H
            0D57  7508            JNZ     @86
505   2                   DO;
506   3                   MODE$8086 = FALSE;
                                                    ; STATEMENT # 506
            0D59  C606730000      MOV     MODE8086,0H
507   3                   CALL SIO$GET$CHAR;
                                                    ; STATEMENT # 507
            0D5E  E8AEF5          CALL    SIOGETCHAR
508   3                   END;
                           @86:
509   2              IF CHAR=ASBL THEN CALL SIO$GET$CHAR;
                                                    ; STATEMENT # 509
            0D61  803E660020      CMP     CHAR,20H
            0D66  7503            JNZ     @87
                                                    ; STATEMENT # 510
            0D68  E8A4F5          CALL    SIOGETCHAR
                           @87:
511   2              CALL SIO$GET$ADDR(@ARG1,CS);
                                                    ; STATEMENT # 511
            0D6B  8D061A00        LEA     AX,ARG1
            0D6F  1E              PUSH    DS      ; 1
            0D70  50              PUSH    AX      ; 2
            0D71  FF364000        PUSH    REGSAV+10H; 3
            0D75  E86CF8          CALL    SIOGETADDR
512   2              IF CHAR<>',' THEN GOTO ERROR;
                                                    ; STATEMENT # 512
            0D78  803E66002C      CMP     CHAR,2CH
            0D7D  7403            JZ      $+5H
            0D7F  E984F4          JMP     ERROR
514   2              CALL SIO$GET$CHAR;
                                                    ; STATEMENT # 514
```

```
                0D82  E88AF5          CALL    SIOGETCHAR
515   2               END$OFF = SIO$GET$WORD;
                                              ; STATEMENT # 515
                0D85  E868F7          CALL    SIOGETWORD
                0D88  A31800          MOV     ENDOFF,AX
516   2               IF END$OFF<ARG1.OFF THEN GOTO ERROR;
                                              ; STATEMENT # 516
                0D8B  A11800          MOV     AX,ENDOFF
                0D8E  3B061A00        CMP     AX,ARG1
                0D92  7303            JNB     $+5H
                0D94  E96FF4          JMP     ERROR
518   2               IF CHAR<>ASCR THEN
                                              ; STATEMENT # 518
                0D97  803E66000D      CMP     CHAR,0DH
                0D9C  7417            JZ      @90
519   2                     DO;
520   3                     START$REC = TRUE;
                                              ; STATEMENT # 520
                0D9E  C6067200FF      MOV     STARTREC,0FFH
521   3                     CALL SIO$GET$CHAR;
                                              ; STATEMENT # 521
                0DA3  E869F5          CALL    SIOGETCHAR
522   3                     CALL SIO$GET$ADDR(@ARG3,CS);
                                              ; STATEMENT # 522
                0DA6  8D061E00        LEA     AX,ARG3
                0DAA  1E              PUSH    DS      ; 1
                0DAB  50              PUSH    AX      ; 2
                0DAC  FF364000        PUSH    REGSAV+10H; 3
                0DB0  E831F8          CALL    SIOGETADDR
523   3                     END;
                                              ; STATEMENT # 523
                0DB3  EB0B            JMP     @91
                            @90:
                        ELSE
524   2                     DO;
525   3                     START$REC = FALSE;
                                              ; STATEMENT # 525
                0DB5  C606720000      MOV     STARTREC,0H
526   3                     ARG3.OFF = 0;
                                              ; STATEMENT # 526
                0DBA  C7061E000000    MOV     ARG3,0H
527   3                     END;
                             @91:
528   2               IF CHAR<>ASCR THEN GOTO ERROR;
                                              ; STATEMENT # 528
                0DC0  803E66000D      CMP     CHAR,0DH
                0DC5  7403            JZ      $+5H
                0DC7  E93CF4          JMP     ERROR
530   2               CALL SIO$CRLF;
                                              ; STATEMENT # 530

                0DCA  E8DAF6          CALL    SIOCRLF
531   2               DO I=1 TO 60;                 /* LEADING NULLS */
                                              ; STATEMENT # 531
                0DCD  C606680001      MOV     I,1H
                            @154:
                0DD2  803E68003C      CMP     I,3CH
```

```
                0DD7  770D            JA      @155
532   3                 CALL SIO$OUT$CHAR(0);
                                                ; STATEMENT # 532
                0DD9  B000            MOV     AL,0H
                0DDB  50              PUSH    AX      ; 1
                0DDC  E810F5          CALL    SIOOUTCHAR
533   3             END;
                                                ; STATEMENT # 533
                0DDF  8006680001      ADD     I,1H
                0DE4  75EC            JNZ     @154
                          @155:
534   2             CALL SIO$CRLF;
                                                ; STATEMENT # 534

                0DE6  E8BEF6          CALL    SIOCRLF
535   2             IF MODE$8086 THEN
                                                ; STATEMENT # 535
                0DE9  A07300          MOV     AL,MODE8086
                0DEC  D0D8            RCR     AL,1
                0DEE  734E            JNB     @93
536   2                 DO;
537   3                 IF START$REC THEN
                                                ; STATEMENT # 537
                0DF0  A07200          MOV     AL,STARTREC
                0DF3  D0D8            RCR     AL,1
                0DF5  732B            JNB     @94
538   3                     DO;
539   4                     CALL SIO$OUT$HEADER(04,0,03); /* START ADDRESS RECORD */
                                                ; STATEMENT # 539
                0DF7  B004            MOV     AL,4H
                0DF9  50              PUSH    AX      ; 1
                0DFA  B80000          MOV     AX,0H
                0DFD  50              PUSH    AX      ; 2
                0DFE  B003            MOV     AL,3H
                0E00  50              PUSH    AX      ; 3
                0E01  E8B1F5          CALL    SIOOUTHEADER
540   4                     CALL SIO$OUT$WORD(ARG3.SEG);
                                                ; STATEMENT # 540
                0E04  FF362000        PUSH    ARG3+2H ; 1
                0E08  E855F5          CALL    SIOOUTWORD
541   4                     CALL SIO$OUT$WORD(ARG3.OFF);
                                                ; STATEMENT # 541
                0E0B  FF361E00        PUSH    ARG3    ; 1
                0E0F  E84EF5          CALL    SIOOUTWORD
542   4                     CALL SIO$OUT$BYTE(CHECK$SUM);
                                                ; STATEMENT # 542
                0E12  FF366700        PUSH    CHECKSUM; 1
                0E16  E817F5          CALL    SIOOUTBYTE
543   4                     CALL SIO$CRLF;
                                                ; STATEMENT # 543
                0E19  E88BF6          CALL    SIOCRLF
544   4                     ARG3.OFF = 0;
                                                ; STATEMENT # 544
                0E1C  C7061E000000    MOV     ARG3,0H
545   4                     END;
                          @94:
546   3                 CALL SIO$OUT$HEADER(02,0,02); /* EXTENDED ADDRESS RECORD */
```

```
                                                    ; STATEMENT # 546
        0E22  B002          MOV     AL,2H
        0E24  50            PUSH    AX      ; 1
        0E25  B90000        MOV     CX,0H
        0E28  51            PUSH    CX      ; 2
        0E29  50            PUSH    AX      ; 3
        0E2A  E888F5        CALL    SIOOUTHEADER
547  3              CALL SIO$OUT$WORD(ARG1.SEG);
                                                    ; STATEMENT # 547
        0E2D  FF361C00      PUSH    ARG1+2H ; 1
        0E31  E82CF5        CALL    SIOOUTWORD
548  3              CALL SIO$OUT$BYTE(CHECK$SUM);
                                                    ; STATEMENT # 548
        0E34  FF366700      PUSH    CHECKSUM; 1
        0E38  E8F5F4        CALL    SIOOUTBYTE
549  3              CALL SIO$CRLF;
                                                    ; STATEMENT # 549
        0E3B  E869F6        CALL    SIOCRLF
550  3              END;
                    @93:

551  2          LEN = STANDARD$LEN;                  /* DATA RECORD */
                                                    ; STATEMENT # 551
        0E3E  C7065C001000  MOV     LEN,10H
552  2      LOOP: INDEX = END$OFF - ARG1.OFF;
                                                    ; STATEMENT # 552
                    LOOP:
        0E44  A11800        MOV     AX,ENDOFF
        0E47  2B061A00      SUB     AX,ARG1
        0E4B  A35E00        MOV     INDEX,AX
553  2          IF INDEX<STANDARD$LEN THEN LEN = INDEX + 1;
                                                    ; STATEMENT # 553
        0E4E  83F810        CMP     AX,10H
        0E51  7309          JNB     @95
                                                    ; STATEMENT # 554
        0E53  A15E00        MOV     AX,INDEX
        0E56  83C001        ADD     AX,1H
        0E59  A35C00        MOV     LEN,AX
                    @95:
555  2          CALL SIO$OUT$HEADER(LEN,ARG1.OFF,00);
                                                    ; STATEMENT # 555
        0E5C  FF365C00      PUSH    LEN     ; 1
        0E60  FF361A00      PUSH    ARG1    ; 2
        0E64  B000          MOV     AL,0H
        0E66  50            PUSH    AX      ; 3
        0E67  E84BF5        CALL    SIOOUTHEADER
556  2          DO I=1 TO LEN;
                                                    ; STATEMENT # 556
        0E6A  C606680001    MOV     I,1H
                    @156:
        0E6F  A06800        MOV     AL,I
        0E72  B400          MOV     AH,0H
        0E74  3B065C00      CMP     AX,LEN
        0E78  7716          JA      @157
557  3              CALL SIO$OUT$BYTE(MEMORY$ARG1);
                                                    ; STATEMENT # 557
        0E7A  C41E1A00      LES     BX,MEMORYARG1PTR
```

```
                0E7E  26FF37          PUSH    ES:MEMORYARG1[BX]
                0E81  E8ACF4          CALL    SIOOUTBYTE
558   3                   ARG1.OFF = ARG1.OFF + 1;
                                              ; STATEMENT # 558
                0E84  83061A0001      ADD     ARG1,1H
559   3               END;
                                              ; STATEMENT # 559
                0E89  8006680001      ADD     I,1H
                0E8E  75DF            JNZ     @156
                            @157:
560   2               CALL SIO$OUT$BYTE(CHECK$SUM);
                                              ; STATEMENT # 560
                0E90  FF366700        PUSH    CHECKSUM; 1
                0E94  E899F4          CALL    SIOOUTBYTE
561   2               CALL SIO$CRLF;
                                              ; STATEMENT # 561
                0E97  E80DF6          CALL    SIOCRLF
562   2               IF END$OFF<>ARG1.OFF-1 THEN GOTO LOOP;
                                              ; STATEMENT # 562
                0E9A  A11A00          MOV     AX,ARG1
                0E9D  83E801          SUB     AX,1H
                0EA0  39061800        CMP     ENDOFF,AX
                0EA4  759E            JNZ     LOOP

564   2               CALL SIO$OUT$HEADER(00,ARG3.OFF,01);     /* EOF RECORD */
                                              ; STATEMENT # 564
                0EA6  B000            MOV     AL,0H
                0EA8  50              PUSH    AX      ; 1
                0EA9  FF361E00        PUSH    ARG3    ; 2
                0EAD  B001            MOV     AL,1H
                0EAF  50              PUSH    AX      ; 3
                0EB0  E802F5          CALL    SIOOUTHEADER
565   2               CALL SIO$OUT$BYTE(CHECK$SUM);
                                              ; STATEMENT # 565
                0EB3  FF366700        PUSH    CHECKSUM; 1
                0EB7  E876F4          CALL    SIOOUTBYTE
566   2               CALL SIO$CRLF;
                                              ; STATEMENT # 566

                0EBA  E8EAF5          CALL    SIOCRLF
567   2               DO I=1 TO 60;           /* TRAILING NULLS */
                                              ; STATEMENT # 567
                0EBD  C606680001      MOV     I,1H
                            @158:
                0EC2  803E68003C      CMP     I,3CH
                0EC7  770D            JA      @159
568   3                   CALL SIO$OUT$CHAR(0);
                                              ; STATEMENT # 568
                0EC9  B000            MOV     AL,0H
                0ECB  50              PUSH    AX      ; 1
                0ECC  E820F4          CALL    SIOOUTCHAR
569   3               END;
                                              ; STATEMENT # 569
                0ECF  8006680001      ADD     I,1H
                0ED4  75EC            JNZ     @158
                            @159:
570   2           END;
```

```
                                                    ; STATEMENT # 570
          0ED6  5D              POP     BP
          0ED7  C3              RET
                        SIOWRITE        ENDP

571   1       SIO$READ:
                                                    ; STATEMENT # 571
                  /* IMPLEMENTS THE READ PAPER TAPE COMMAND. */
                        SIOREAD         PROC NEAR
          0ED8  55              PUSH    BP
          0ED9  8BEC            MOV     BP,SP
                  PROCEDURE;
572   2               DECLARE BIAS WORD, (REC$TYPE,LEN,I,T) BYTE, OFFSET WORD;
573   2               BIAS = 0;                     /* DEFAULT BIAS */
                                                    ; STATEMENT # 573
          0EDB  C70660000000    MOV     BIAS,0H
574   2               ARG1.SEG = 0;                 /* DEFAULT CS FOR 8080 FORMAT FILE */
                                                    ; STATEMENT # 574
          0EE1  C7061C000000    MOV     ARG1+2H,0H
575   2               CALL SIO$SCAN$BLANK;
                                                    ; STATEMENT # 575
          0EE7  E8F4F5          CALL    SIOSCANBLANK
576   2               IF CHAR<>ASCR THEN  BIAS = SIO$GET$WORD;
                                                    ; STATEMENT # 576
          0EEA  803E66000D      CMP     CHAR,0DH
          0EEF  7406            JZ      @97
                                                    ; STATEMENT # 577
          0EF1  E8FCF5          CALL    SIOGETWORD
          0EF4  A36000          MOV     BIAS,AX
                        @97:
578   2               IF CHAR<>ASCR THEN GOTO ERROR;
                                                    ; STATEMENT # 578
          0EF7  803E66000D      CMP     CHAR,0DH
          0EFC  7403            JZ      $+5H
          0EFE  E905F3          JMP     ERROR
580   2               CALL SIO$CRLF;
                                                    ; STATEMENT # 580
          0F01  E8A3F5          CALL    SIOCRLF
581   2           LOOP:
                                                    ; STATEMENT # 581
                        LOOP:
          0F04  E87BF7          CALL    SIOREADCHAR
          0F07  3C3A            CMP     AL,3AH
          0F09  75F9            JNZ     LOOP
                  DO WHILE SIO$READ$CHAR<>':';END;
583   2               CHECK$SUM = 0;
                                                    ; STATEMENT # 583
          0F0B  C606670000      MOV     CHECKSUM,0H
584   2               LEN = SIO$READ$BYTE;
                                                    ; STATEMENT # 584
          0F10  E8BAF7          CALL    SIOREADBYTE
          0F13  A27500          MOV     LEN,AL
585   2               OFFSET = SIO$READ$WORD;
                                                    ; STATEMENT # 585
          0F16  E8DFF7          CALL    SIOREADWORD
          0F19  A36200          MOV     OFFSET,AX
586   2               ARG1.OFF = OFFSET + BIAS;
```

```
                                                    ; STATEMENT # 586
        0F1C  A16200          MOV     AX,OFFSET
        0F1F  03066000        ADD     AX,BIAS
        0F23  A31A00          MOV     ARG1,AX
587   2             REC$TYPE = SIO$READ$BYTE;
                                                    ; STATEMENT # 587
        0F26  E8A4F7          CALL    SIOREADBYTE
        0F29  A27400          MOV     RECTYPE,AL
588   2             IF REC$TYPE=03 THEN       /* START ADDR TYPE */
                                                    ; STATEMENT # 588
        0F2C  803E740003      CMP     RECTYPE,3H
        0F31  750C            JNZ     @99
589   2                 DO;
590   3                 CS = SIO$READ$WORD;
                                                    ; STATEMENT # 590
        0F33  E8C2F7          CALL    SIOREADWORD
        0F36  A34000          MOV     REGSAV+10H,AX
591   3                 IP = SIO$READ$WORD;
                                                    ; STATEMENT # 591
        0F39  E8BCF7          CALL    SIOREADWORD
        0F3C  A34800          MOV     REGSAV+18H,AX
592   3                 END;
                           @99:
593   2             IF REC$TYPE=02 THEN       /* EXTENDED ADDR TYPE */
                                                    ; STATEMENT # 593
        0F3F  803E740002      CMP     RECTYPE,2H
        0F44  7506            JNZ     @100
594   2                 ARG1.SEG = SIO$READ$WORD;
                                                    ; STATEMENT # 594
        0F46  E8AFF7          CALL    SIOREADWORD
        0F49  A31C00          MOV     ARG1+2H,AX
                           @100:
595   2             IF REC$TYPE = 01 THEN     /* EOF */
                                                    ; STATEMENT # 595
        0F4C  803E740001      CMP     RECTYPE,1H
        0F51  750D            JNZ     @101
596   2                 IF OFFSET <> 0 THEN IP = OFFSET;
                                                    ; STATEMENT # 596
        0F53  833E620000      CMP     OFFSET,0H
        0F58  7406            JZ      @101
                                                    ; STATEMENT # 597
        0F5A  A16200          MOV     AX,OFFSET
        0F5D  A34800          MOV     REGSAV+18H,AX
                           @101:
598   2             IF REC$TYPE=00 THEN       /* DATA TYPE */
                                                    ; STATEMENT # 598
        0F60  823E740000      CMP     RECTYPE,0H
        0F65  7537            JNZ     @103
599   2                 DO I=1 TO LEN;
                                                    ; STATEMENT # 599
        0F67  C606760001      MOV     I,1H
                           @162:
        0F6C  A07600          MOV     AL,I
        0F6F  3A067500        CMP     AL,LEN
        0F73  7729            JA      @103
600   3                       T,MEMORY$ARG1 = SIO$READ$BYTE;
                                                    ; STATEMENT # 600
```

```
                0F75  E855F7          CALL    SIOREADBYTE
                0F78  A27700          MOV     T,AL
                0F7B  C41E1A00        LES     BX,MEMORYARG1PTR
                0F7F  268807          MOV     ES:MEMORYARG1[BX],AL
601     3                       IF MEMORY$ARG1<>T THEN GOTO ERROR;
                                                      ; STATEMENT # 601
                0F82  C41E1A00        LES     BX,MEMORYARG1PTR
                0F86  268A07          MOV     AL,ES:MEMORYARG1[BX]
                0F89  3A067700        CMP     AL,T
                0F8D  7403            JZ      $+5H
                0F8F  E974F2          JMP     ERROR
603     3                       ARG1.OFF = ARG1.OFF + 1;
                                                      ; STATEMENT # 603
                0F92  83061A0001      ADD     ARG1,1H
604     3                     END;
                                                      ; STATEMENT # 604
                0F97  8006760001      ADD     I,1H
                0F9C  75CE            JNZ     @162
                                @103:
605     2                   T = SIO$READ$BYTE;        /* FETCH CHECKSUM */
                                                      ; STATEMENT # 605
                0F9E  E82CF7          CALL    SIOREADBYTE
                0FA1  A27700          MOV     T,AL
606     2                   IF CHECK$SUM<>0 THEN GOTO ERROR;
                                                      ; STATEMENT # 606
                0FA4  823E670000      CMP     CHECKSUM,0H
                0FA9  7403            JZ      $+5H
                0FAB  E958F2          JMP     ERROR
608     2                   IF REC$TYPE<>01 AND LEN<>0 THEN GOTO LOOP;
                                                      ; STATEMENT # 608
                0FAE  803E740001      CMP     RECTYPE,1H
                0FB3  B0FF            MOV     AL,0FFH
                0FB5  7501            JNZ     $+3H
                0FB7  40              INC     AX
                0FB8  50              PUSH    AX      ; 1
                0FB9  823E750000      CMP     LEN,0H
                0FBE  B0FF            MOV     AL,0FFH
                0FC0  7501            JNZ     $+3H
                0FC2  40              INC     AX
                0FC3  59              POP     CX      ; 1
                0FC4  22C1            AND     AL,CL
                0FC6  D0D8            RCR     AL,1
                0FC8  7303            JNB     $+5H
                0FCA  E937FF          JMP     LOOP
610     2                   CALL SIO$OUT$CHAR(0);  /* DELAY FOR LAST CR, LF SENT */
                                                      ; STATEMENT # 610
                0FCD  B000            MOV     AL,0H
                0FCF  50              PUSH    AX      ; 1
                0FD0  E81CF3          CALL    SIOOUTCHAR
611     2                   CALL SIO$OUT$CHAR(0);  /*    BY INTELLEC */
                                                      ; STATEMENT # 611
                0FD3  B000            MOV     AL,0H
                0FD5  50              PUSH    AX      ; 1
                0FD6  E816F3          CALL    SIOOUTCHAR
612     2                    END;
                                                      ; STATEMENT # 612
                0FD9  5D              POP     BP
```

```
        0FDA  C3              RET
                     SIOREAD        ENDP

              /******************************************
              *   COMMAND DISPATCH MAIN PROGRAM LOOP    *
              ******************************************/

613   1             DISABLE;
                                                 ; STATEMENT # 613
        00A8  FA              CLI
        00A9  2E8E16A400      MOV     SS,CS:@@STACK$FRAME
        00AE  BC4200          MOV     SP,@@STACK$OFFSET
        00B1  8BEC            MOV     BP,SP
        00B3  2E8E1EA600      MOV     DS,CS:@@DATA$FRAME
        00B8  FB              STI
        00B9  FA              CLI
614   1             OUTPUT(KB$STAT$PORT) = KB$INIT$MODE;
                                                 ; STATEMENT # 614
        00BA  BAEAFF          MOV     DX,0FFEAH
        00BD  B000            MOV     AL,0H
        00BF  EE              OUT     DX
615   1             OUTPUT(KB$STAT$PORT) = KB$INIT$SCAN;
                                                 ; STATEMENT # 615
        00C0  B039            MOV     AL,39H
        00C2  EE              OUT     DX
616   1             OUTPUT(KB$STAT$PORT) = 90H; /* OUTPUT '8086' SIGNON BACKWARDS */
                                                 ; STATEMENT # 616
        00C3  B090            MOV     AL,90H
        00C5  EE              OUT     DX
617   1             DO I=0 TO 7;
                                                 ; STATEMENT # 617
        00C6  C606680000      MOV     I,0H
                     @164:
        00CB  803E680007      CMP     I,7H
        00D0  7715            JA      @165
618   2               OUTPUT(KB$DATA$PORT) = KB$SIGNON(I);
                                                 ; STATEMENT # 618
        00D2  8A1E6800        MOV     BL,I
        00D6  B700            MOV     BH,0H
        00D8  2E8A471B        MOV     AL,CS:KBSIGNON[BX]
        00DC  BAE8FF          MOV     DX,0FFE8H
        00DF  EE              OUT     DX
619   2             END;
                                                 ; STATEMENT # 619
        00E0  8006680001      ADD     I,1H
        00E5  75E4            JNZ     @164
                     @165:

              /* THIS STRANGE SEQUENCE OF CODE IS USED TO GUARANTEE CORRECT USART
                 INITIALIZATION AFTER EITHER: 1) HARDWARE RESET (EXPECTING MODE)
                 2) SOFTWARE RESTART (EXPECTING COMMAND). */

620   1             OUTPUT(SIO$STAT$PORT) = SIO$8251$RESET;
                                                 ; STATEMENT # 620
        00E7  BAF2FF          MOV     DX,0FFF2H
        00EA  B065            MOV     AL,65H
```

```
                00EC  EE              OUT     DX
621   1               CALL SIO$8251$SETTLING$DELAY;
                                              ; STATEMENT # 621
                00ED  E8E902          CALL    SIO8251SETTLINGDELAY
622   1               OUTPUT(SIO$STAT$PORT) = 25H;
                                              ; STATEMENT # 622
                00F0  BAF2FF          MOV     DX,0FFF2H
                00F3  B025            MOV     AL,25H
                00F5  EE              OUT     DX
623   1               CALL SIO$8251$SETTLING$DELAY;
                                              ; STATEMENT # 623
                00F6  E8E002          CALL    SIO8251SETTLINGDELAY
624   1               OUTPUT(SIO$STAT$PORT) = SIO$8251$RESET;
                                              ; STATEMENT # 624
                00F9  BAF2FF          MOV     DX,0FFF2H
                00FC  B065            MOV     AL,65H
                00FE  EE              OUT     DX
625   1               CALL SIO$8251$SETTLING$DELAY;
                                              ; STATEMENT # 625
                00FF  E8D702          CALL    SIO8251SETTLINGDELAY
626   1               OUTPUT(SIO$STAT$PORT) = SIO$8251$MODE;
                                              ; STATEMENT # 626
                0102  BAF2FF          MOV     DX,0FFF2H
                0105  B0CF            MOV     AL,0CFH
                0107  EE              OUT     DX
627   1               CALL SIO$8251$SETTLING$DELAY;
                                              ; STATEMENT # 627
                0108  E8CE02          CALL    SIO8251SETTLINGDELAY
628   1               OUTPUT(SIO$STAT$PORT) = SIO$8251$CMND;
                                              ; STATEMENT # 628
                010B  BAF2FF          MOV     DX,0FFF2H
                010E  B025            MOV     AL,25H
                0110  EE              OUT     DX
629   1               CALL SIO$8251$SETTLING$DELAY;
                                              ; STATEMENT # 629
                0111  E8C502          CALL    SIO8251SETTLINGDELAY
630   1               CALL SIO$OUT$STRING(@SIO$SIGNON);
                                              ; STATEMENT # 630
                0114  2E8D062700      LEA     AX,CS:SIOSIGNON
                0119  0E              PUSH    CS      ; 1
                011A  50              PUSH    AX      ; 2
                011B  E86402          CALL    SIOOUTSTRING

                    /* INITIALIZE USER'S REGISTERS */
631   1               CS,SS,DS,ES,FL,IP = 0;
                                              ; STATEMENT # 631
                011E  B80000          MOV     AX,0H
                0121  A34000          MOV     REGSAV+10H,AX
                0124  A34400          MOV     REGSAV+14H,AX
                0127  A34200          MOV     REGSAV+12H,AX
                012A  A34600          MOV     REGSAV+16H,AX
                012D  A34A00          MOV     REGSAV+1AH,AX
                0130  A34800          MOV     REGSAV+18H,AX
632   1               SP = USER$INIT$SP;
                                              ; STATEMENT # 632

                    /* INITIALIZE INTERRUPT VECTORS */
```

```
        0133  C70638000001    MOV     REGSAV+8H,100H
633   1           CALL INIT$INT$VECTOR(@INT$VECTOR(1),.INTERRUPT1$ENTRY);
                                        ; STATEMENT # 633
        0139  8D060400        LEA     AX,INTVECTOR+4H
        013D  1E              PUSH    DS      ; 1
        013E  50              PUSH    AX      ; 2
        013F  B8DF07          MOV     AX,OFFSET(INTERRUPT1ENTRY)
        0142  50              PUSH    AX      ; 3
        0143  E82B07          CALL    INITINTVECTOR
634   1           CALL INIT$INT$VECTOR(@INT$VECTOR(2),.INTERRUPT3$ENTRY);
                                        ; STATEMENT # 634
        0146  8D060800        LEA     AX,INTVECTOR+8H
        014A  1E              PUSH    DS      ; 1
        014B  50              PUSH    AX      ; 2
        014C  B85108          MOV     AX,OFFSET(INTERRUPT3ENTRY)
        014F  50              PUSH    AX      ; 3
        0150  E81E07          CALL    INITINTVECTOR
635   1           CALL INIT$INT$VECTOR(@INT$VECTOR(3),.INTERRUPT3$ENTRY);
                                        ; STATEMENT # 635
        0153  8D060C00        LEA     AX,INTVECTOR+0CH
        0157  1E              PUSH    DS      ; 1
        0158  50              PUSH    AX      ; 2
        0159  B85108          MOV     AX,OFFSET(INTERRUPT3ENTRY)
        015C  50              PUSH    AX      ; 3
        015D  E81107          CALL    INITINTVECTOR

636   1           BRK1$FLAG = FALSE;
                                        ; STATEMENT # 636
        0160  C606640000      MOV     BRK1FLAG,0H
637   1           MONITOR$STACKPTR = STACKPTR;
                                        ; STATEMENT # 637
        0165  89E0            MOV     AX,SP
        0167  A31400          MOV     MONITORSTACKPTR,AX
638   1           MONITOR$STACKBASE = STACKBASE;
                                        ; STATEMENT # 638

        016A  8CD0            MOV     AX,SS
        016C  A31600          MOV     MONITORSTACKBASE,AX
639   1       NEXT$COMMAND:
                                        ; STATEMENT # 639
              /* THIS IS THE PERPETUAL COMMAND LOOP WHICH DISPATCHES TO EACH
                 COMMAND WHICH IS A SEPARATED PROCEDURE. */

                  NEXTCOMMAND:
        016F  BC4200          MOV     SP,@@STACK$OFFSET
        0172  8BEC            MOV     BP,SP
        0174  2E8E1EA600      MOV     DS,CS:@@DATA$FRAME
                  CALL SIO$CRLF;
        0179  E82B03          CALL    SIOCRLF
640   1           CALL SIO$OUT$CHAR(0);
                                        ; STATEMENT # 640
        017C  B000            MOV     AL,0H
        017E  50              PUSH    AX      ; 1
        017F  E86D01          CALL    SIOOUTCHAR
641   1           CALL SIO$OUT$CHAR('.');
                                        ; STATEMENT # 641
        0182  B02E            MOV     AL,2EH
```

```
                0184  50              PUSH    AX      ; 1
                0185  E86701          CALL    SIOOUTCHAR
642   1             CALL SIO$GET$CHAR;
                                                      ; STATEMENT # 642
                0188  E88401          CALL    SIOGETCHAR
643   1             DO I=0 TO LAST(SIO$CMND);
                                                      ; STATEMENT # 643
                018B  C606680000      MOV     I,0H
                            @166:
                0190  803E680009      CMP     I,9H
                0195  776F            JA      ERROR
644   2                 IF CHAR=SIO$CMND(I) THEN GOTO DISPATCH;
                                                      ; STATEMENT # 644
                0197  A06600          MOV     AL,CHAR
                019A  8A1E6800        MOV     BL,I
                019E  B700            MOV     BH,0H
                01A0  2E3A474E        CMP     AL,CS:SIOCMND[BX]
                01A4  7409            JZ      DISPATCH
646   2             END;
                                                      ; STATEMENT # 646
                01A6  8006680001      ADD     I,1H
                01AB  75E3            JNZ     @166
647   1             GOTO ERROR;
                                                      ; STATEMENT # 647
                01AD  EB57            JMP     ERROR
648   1         DISPATCH:
                                                      ; STATEMENT # 648
                        DISPATCH:
                    LAST$COMMAND = I;
                01AF  A06800          MOV     AL,I
                01B2  A26A00          MOV     LASTCOMMAND,AL
649   1             DO CASE I;
                                                      ; STATEMENT # 649
                01B5  B400            MOV     AH,0H
                01B7  89C3            MOV     BX,AX
                01B9  D1E3            SLL     BX,1
                01BB  2EFFA7F201      JMP     CS:@168[BX]
650   2                 CALL SIO$EXAM$MEM;
                                                      ; STATEMENT # 650
                            @109:
                01C0  E86307          CALL    SIOEXAMMEM
                01C3  EBAA            JMP     NEXTCOMMAND
651   2                 CALL SIO$EXAM$REG;
                                                      ; STATEMENT # 651
                            @110:
                01C5  E83E08          CALL    SIOEXAMREG
                01C8  EBA5            JMP     NEXTCOMMAND
652   2                 CALL SIO$GO;
                                                      ; STATEMENT # 652
                            @111:
                01CA  E8C506          CALL    SIOGO
                01CD  EBA0            JMP     NEXTCOMMAND
653   2                 CALL SIO$SINGLE$STEP;
                                                      ; STATEMENT # 653
                            @112:
                01CF  E82D07          CALL    SIOSINGLESTEP
                01D2  EB9B            JMP     NEXTCOMMAND
```

```
654   2                 CALL SIO$MOVE;
                                                    ; STATEMENT # 654
                        @113:
          01D4  E87E09            CALL    SIOMOVE
          01D7  EB96              JMP     NEXTCOMMAND
655   2                 CALL SIO$DISPLAY;
                                                    ; STATEMENT # 655
                        @114:
          01D9  E80A0A            CALL    SIODISPLAY
          01DC  EB91              JMP     NEXTCOMMAND
656   2                 CALL SIO$INPUT;
                                                    ; STATEMENT # 656
                        @115:
          01DE  E8C50A            CALL    SIOINPUT
          01E1  EB8C              JMP     NEXTCOMMAND
657   2                 CALL SIO$OUTPUT;
                                                    ; STATEMENT # 657
                        @116:
          01E3  E8000B            CALL    SIOOUTPUT
          01E6  EB87              JMP     NEXTCOMMAND
658   2                 CALL SIO$READ;
                                                    ; STATEMENT # 658
                        @117:
          01E8  E8ED0C            CALL    SIOREAD
                        @12:
          01EB  EB82              JMP     NEXTCOMMAND
659   2                 CALL SIO$WRITE;
                                                    ; STATEMENT # 659
                        @118:
          01ED  E8570B            CALL    SIOWRITE
          01F0  EBF9              JMP     @12
660   2             END;
                                                    ; STATEMENT # 660
                        @168:
          01F2  C001              DW      @109
          01F4  C501              DW      @110
          01F6  CA01              DW      @111
          01F8  CF01              DW      @112
          01FA  D401              DW      @113
          01FC  D901              DW      @114
          01FE  DE01              DW      @115
          0200  E301              DW      @116
          0202  E801              DW      @117
          0204  ED01              DW      @118
661   1             GOTO NEXT$COMMAND;

662   1       ERROR:
                                                    ; STATEMENT # 662
                  /* THIS ROUTINE HANDLES ALL ERRORS DETECTED BY THE MONITOR AND
                     WILL OUTPUT THE ERROR PROMPT TO THE OUTPUT PORT.*/

                        ERROR:
          0206  BC4200            MOV     SP,@@STACK$OFFSET
          0209  8BEC              MOV     BP,SP
          020B  2E8E1EA600        MOV     DS,CS:@@DATA$FRAME
          0210  B023              MOV     AL,23H
          0212  50                PUSH    AX      ; 1
```

```
          0213  E8D900          CALL    SIOOUTCHAR
                    CALL SIO$OUT$CHAR('#');
663   1             GOTO NEXT$COMMAND;
                                        ; STATEMENT # 663
          0216  EBD3            JMP     @12

664   1       AFTER$INTERRUPT:
                                        ; STATEMENT # 664
                 /* THIS ROUTINE IS CALLED AFTER AN INTERRUPT TO DISPLAY THE CS:IP
                    AND RESTORE THE BREAKPOINTED INSTRUCTION. */

                      AFTERINTERRUPT:
          0218  BC4200          MOV     SP,@@STACK$OFFSET
          021B  8BEC            MOV     BP,SP
          021D  2E8E1EA600      MOV     DS,CS:@@DATA$FRAME
          0222  A06400          MOV     AL,BRK1FLAG
          0225  D0D8            RCR     AL,1
          0227  735A            JNB     @119
                    IF BRK1$FLAG THEN
665   1                DO;
666   2                MEMORY$BRK1 = BRK1$SAVE;
                                        ; STATEMENT # 666
          0229  A06500          MOV     AL,BRK1SAVE
          022C  C41E2200        LES     BX,MEMORYBRK1PTR
          0230  268807          MOV     ES:MEMORYBRK1[BX],AL
667   2                BRK1$FLAG = FALSE;
                                        ; STATEMENT # 667
          0233  C606640000      MOV     BRK1FLAG,0H
668   2                IF ((IP-1) AND 000FH)=(BRK1.OFF AND 000FH) AND
                                        ; STATEMENT # 668
          0238  A14800          MOV     AX,REGSAV+18H
          023B  83E801          SUB     AX,1H
          023E  50              PUSH    AX      ; 1
          023F  250F00          AND     AX,0FH
          0242  8B0E2200        MOV     CX,BRK1
          0246  51              PUSH    CX      ; 2
          0247  81E10F00        AND     CX,0FH
          024B  3BC1            CMP     AX,CX
          024D  B0FF            MOV     AL,0FFH
          024F  7401            JZ      $+3H
          0251  40              INC     AX
          0252  5A              POP     DX      ; 2
          0253  8756FE          XCHG    DX,[BP]-2H; 1
          0256  B104            MOV     CL,4H
          0258  D3EA            SHR     DX,CL
          025A  03164000        ADD     DX,REGSAV+10H
          025E  5B              POP     BX      ; 1
          025F  D3EB            SHR     BX,CL
          0261  031E2400        ADD     BX,BRK1+2H
          0265  50              PUSH    AX      ; 1
          0266  3BD3            CMP     DX,BX
          0268  B0FF            MOV     AL,0FFH
          026A  7401            JZ      $+3H
          026C  40              INC     AX
          026D  59              POP     CX      ; 1
          026E  22C1            AND     AL,CL
          0270  D0D8            RCR     AL,1
```

```
          0272  730F            JNB     @119
                              (SHR(IP-1,4)+CS)=(SHR(BRK1.OFF,4)+BRK1.SEG) THEN
669   2                     DO;
670   3                     IP = IP - 1;
                                                    ; STATEMENT # 670
          0274  832E480001        SUB     REGSAV+18H,1H
671   3                     CALL SIO$OUT$STRING(@SIO$BREAK$MSG);
                                                    ; STATEMENT # 671
          0279  2E8D062300        LEA     AX,CS:SIOBREAKMSG
          027E  0E                PUSH    CS      ; 1
          027F  50                PUSH    AX      ; 2
          0280  E8FF00            CALL    SIOOUTSTRING
672   3                     END;
673   2                   END;
                           @119:
674   1               CALL SIO$OUT$CHAR('@');
                                                    ; STATEMENT # 674
          0283  B040              MOV     AL,40H
          0285  50                PUSH    AX      ; 1
          0286  E86600            CALL    SIOOUTCHAR
675   1               CALL SIO$OUT$WORD(CS);
                                                    ; STATEMENT # 675
          0289  FF364000          PUSH    REGSAV+10H; 1
          028D  E8D000            CALL    SIOOUTWORD
676   1               CALL SIO$OUT$CHAR(':');
                                                    ; STATEMENT # 676
          0290  B03A              MOV     AL,3AH
          0292  50                PUSH    AX      ; 1
          0293  E85900            CALL    SIOOUTCHAR
677   1               CALL SIO$OUT$WORD(IP);
                                                    ; STATEMENT # 677
          0296  FF364800          PUSH    REGSAV+18H; 1
          029A  E8C300            CALL    SIOOUTWORD
678   1               GOTO NEXT$COMMAND;
                                                    ; STATEMENT # 678
          029D  E9CFFE            JMP     NEXTCOMMAND

679   1             END MONITOR;        /* END OF MODULE */
              EOF
```

CROSS-REFERENCE LISTING

DEFN	ADDR	SIZE	NAME, ATTRIBUTES, AND REFERENCES	
664	0218H		AFTERINTERRUPT	LABEL 277
169	0000H	4	ARG	STRUCTURE BASED(PTR) 170 171 175 176
11	001AH	4	ARG1	STRUCTURE AT 318 351 352 354 404 409 421 423 430 432 438 444 449 451 454 456 457 511 516 547 552 555 558 562 574 586 594 603
11	001EH	4	ARG3	STRUCTURE AT 414 424 522 526 540 541 544 564
10			ASBL	LITERALLY 48 63 125 130 509
7	003EH	16	ASCII	BYTE ARRAY(16) DATA 53 54 86 87
10			ASCR	LITERALLY 114 160 191 266 290 298 329 334 348 360 387 390 394 415 431 440 473 496 518 528 576 578
10			ASLF	LITERALLY 115
51	0004H	1	B	BYTE PARAMETER AUTOMATIC 52 53 54 55
572	0060H	2	BIAS	WORD 573 577 586
15	FHFFF6H	2	BLOCK8089	WORD AT INITIAL ABSOLUTE
15	FHFFF8H	4	BLOCK8089PTR	POINTER AT INITIAL ABSOLUTE
13	FHFFF0H	1	BOOT1	BYTE ARRAY(1) AT INITIAL ABSOLUTE
13	FHFFF1H	2	BOOT2	WORD ARRAY(1) AT INITIAL ABSOLUTE
13	FHFFF3H	2	BOOT3	WORD ARRAY(1) AT INITIAL ABSOLUTE
14	0000H	4	BOOT4	WORD ARRAY(2) DATA
12			BP	LITERALLY 245

```
     6                  BREAKINST           LITERALLY
                                             293  294

    11  0022H      4   BRK1                STRUCTURE AT
                                             289  668

     5  0064H      1   BRK1FLAG            BYTE
                                             296  636  664  667

     5  0065H      1   BRK1SAVE            BYTE
                                             292  666

    92  0004H      1   C                   BYTE PARAMETER AUTOMATIC
                                              93   94   95   96

    36  0004H      1   C                   BYTE PARAMETER AUTOMATIC
                                              37   42

   105  0006H      1   C1                  BYTE PARAMETER AUTOMATIC
                                             106  108

   105  0004H      1   C2                  BYTE PARAMETER AUTOMATIC
                                             106  108

     5  0066H      1   CHAR                BYTE
                                              23   24   25   28   29   33   47   48   49
                                             100  120  125  130  138  143  153  154  160
                                             162  163  172  177  191  210  212  259  266
                                             286  290  298  308  319  329  331  334  348
                                             360  376  378  387  390  394  405  411  415
                                             431  434  440  466  473  481  485  490  496
                                             504  509  512  518  528  576  578  644

     5  0067H      1   CHECKSUM            BYTE
                                              55   76  218  542  548  560  565  583  606

     4  0008H     19   COPYRIGHT           BYTE ARRAY(19) DATA

   280                  CORRECTION          LITERALLY
                                             282

    12                  CS                  LITERALLY
                                             184  192  318  404  414  430  511  522  668
                                             675

    11  0026H      4   CSIP                STRUCTURE AT
                                             184  185  192  261  262  289  301  302  310
                                             311

   478  0058H      2   DATUM               WORD
                                             484  488  489

   168  0004H      2   DEFAULTBASE         WORD PARAMETER AUTOMATIC
                                             169  170

   195  0050H      2   DELAY               WORD
                                             197  199  200
```

```
  648  01AFH          DISPATCH                LABEL
                                                645

                      DOUBLE                  BUILTIN
                                                95   96  224

   12                 DS                      LITERALLY
                                                631

    5  0018H       2  ENDOFF                  WORD
                                                408  409  421  432  437  438  449  454  515
                                                516  552  562

  662  0206H          ERROR                   LABEL
                                                 30   34  152  164  178  203  267  291  295
                                                299  309  320  335  340  345  375  379  391
                                                406  410  412  416  420  435  439  441  467
                                                482  486  513  517  529  579  602  607  647

   12                 ES                      LITERALLY
                                                631

  157  0581H          EVAL                    LABEL
                                                147

   16                 F                       LITERALLY
                                                 16

    6                 FALSE                   LITERALLY
                                                 19   90  103  111  118  506  525  636  667

   16  FHFFDBH     5  FILL1                   BYTE ARRAY(5) AT INITIAL ABSOLUTE

   16  FHFFE0H    16  FILL2                   BYTE ARRAY(16) AT INITIAL ABSOLUTE

   16  FHFFF5H     1  FILL3                   BYTE ARRAY(1) AT INITIAL ABSOLUTE

   16  FHFFFCH     4  FILL4                   BYTE ARRAY(4) AT INITIAL ABSOLUTE

   12                 FL                      LITERALLY
                                                254  263  303  312  631

    6                 GOCOMMAND               LITERALLY

   84  0004H       1  H                       BYTE PARAMETER AUTOMATIC
                                                 85   87

                      HIGH                    BUILTIN
                                                 59

  358  0070H       1  I                       BYTE
                                                363  365  366  368  369  380  383  392  394
                                                396  398  399

  572  0076H       1  I                       BYTE
                                                599
```

```
    5  0068H     1  I                        BYTE
                                               67   68   69   70   82   86   87   99  100
                                              229  230  240  242  531  556  567  617  618
                                              643  644  648  649

  501  005EH     2  INDEX                    WORD
                                              552  553  554

  279  0871H    33  INITINTVECTOR            PROCEDURE STACK=000EH
                                              633  634  635

  280  0095H    15  INITINTVECTORCODE        BYTE ARRAY(15) DATA
                                              280

  280  0006H     2  INITINTVECTORCODEPTR     WORD DATA
                                              281

                    INPUT                    BUILTIN
                                               18   23   28   40   45   47  198  207  471

  248  07DFH   114  INTERRUPT1ENTRY          PROCEDURE INTERRUPT(1) STACK=0026H
                                              633

  270  0851H    57  INTERRUPT3ENTRY          PROCEDURE INTERRUPT(3) STACK=0014H
                                              634  635

  279  0004H     2  INTROUTINEOFFSET         WORD PARAMETER AUTOMATIC
                                              280  282

    2  0000H    20  INTVECTOR                POINTER ARRAY(5)
                                              633  634  635

  279  0006H     4  INTVECTORPTR             POINTER PARAMETER AUTOMATIC
                                              280  281  282

                    INWORD                   BUILTIN
                                              470

   12               IP                       LITERALLY
                                              183  185  309  631  668  670  677

    9               KBDATAPORT               LITERALLY
                                              618

    9               KBINITMODE               LITERALLY
                                              614

    9               KBINITSCAN               LITERALLY
                                              615

    7  001BH     8  KBSIGNON                 BYTE ARRAY(8) DATA
                                              618

    9               KBSTATPORT               LITERALLY
                                              614  615  616

                    LAST                     BUILTIN
```

```
                                                86  643

     5  006AH     1  LASTCOMMAND                BYTE
                                                255  648

   501  005CH     2  LEN                        WORD
                                                551  554  555  556

   572  0075H     1  LEN                        BYTE
                                                584  599  608

    73  0008H     1  LENGTH                     BYTE PARAMETER AUTOMATIC
                                                74   77

    73  0006H     2  LOADADDR                   WORD PARAMETER AUTOMATIC
                                                74   78

   418  0BAEH        LOOP                       LABEL
                                                425

   552  0E44H        LOOP                       LABEL
                                                563

   196  0685H        LOOP                       LABEL
                                                211

   466  0CB2H        LOOP                       LABEL
                                                475

   445  0C3BH        LOOP                       LABEL
                                                460

   484  0CFFH        LOOP                       LABEL
                                                497

   581  0F04H        LOOP                       LABEL
                                                609

                     LOW                        BUILTIN
                                                60  216  217  343  344  489

     6              MAXDELAY                   LITERALLY
                                                200

    11  0000H     1  MEMORYARG1                 BYTE BASED(MEMORYARG1PTR)
                                                325  343  344  418  419  453  557  600  601

    11  001AH     4  MEMORYARG1PTR              POINTER
                                                11  324  325  338  339  343  344  418  419
                                                448  453  557  600  601

    11  0000H     1  MEMORYARG3                 BYTE BASED(MEMORYARG3PTR)
                                                418  419

    11  001EH     4  MEMORYARG3PTR              POINTER
                                                11  418  419
```

```
  11  0000H     1  MEMORYBRK1              BYTE BASED(MEMORYBRK1PTR)
                                             292  293  294  666

  11  0022H     4  MEMORYBRK1PTR           POINTER
                                               11  292  293  294  666

  11  0000H     1  MEMORYCSIP              BYTE BASED(MEMORYCSIPPTR)
                                             188

  11  0026H     4  MEMORYCSIPPTR           POINTER
                                               11  188

  11  0000H     2  MEMORYUSERSTACK         WORD BASED(MEMORYUSERSTACKPTR)
                                             227  230  242  245

  11  002AH     4  MEMORYUSERSTACKPTR      POINTER
                                               11  227  230  242  245

  11  0000H     2  MEMORYWORDARG1          WORD BASED(MEMORYARG1PTR)
                                             324  338  339  448

 501  0073H     1  MODE8086                BYTE
                                             503  506  535

   1  00A8H   504  MONITOR                 PROCEDURE STACK=0042H

   3  0016H     2  MONITORSTACKBASE        WORD
                                             252  274  638

   3  0014H     2  MONITORSTACKPTR         WORD
                                             251  273  637

 443  0C31H        NEWLINE                 LABEL
                                             459

 639  016FH        NEXTCOMMAND             LABEL
                                             268  661  663  678

  11  0000H     2  OFF                     WORD MEMBER(CSIP)
                                             185  261  301  310

  11  0000H     2  OFF                     WORD MEMBER(USERSTACK)
                                             228  231  234  239  241  244  246  249  271

  11  0000H     2  OFF                     WORD MEMBER(BRK1)
                                             668

  11  0000H     2  OFF                     WORD MEMBER(ARG3)
                                             424  526  541  544  564

  11  0000H     2  OFF                     WORD MEMBER(ARG1)
                                             351  352  354  409  421  423  432  438  444
                                             449  451  454  456  457  516  552  555  558
                                             562  586  603

 169  0000H     2  OFF                     WORD MEMBER(ARG)
                                             171  175  176
```

```
  280  0000H     2  OFF                    WORD MEMBER(VECTOR)
                                             282

  572  0062H     2  OFFSET                 WORD
                                             585  586  596  597

  134  006BH     1  OPER                   BYTE
                                             135  157  163

                    OUTPUT                 BUILTIN
                                               42  196  202  206  489  614  615  616  618
                                             620  622  624  626  628

                    OUTWORD                BUILTIN
                                             488

  478  005AH     2  PORT                   WORD
                                             480  488  489

  463  0056H     2  PORT                   WORD
                                             465  470  471

  168  0006H     4  PTR                    POINTER PARAMETER AUTOMATIC
                                             169  170  171  175  176

   65  0004H     4  PTR                    POINTER PARAMETER AUTOMATIC
                                              66   68   69

  572  0074H     1  RECTYPE                BYTE
                                             587  588  593  595  598  608

   73  0004H     1  RECTYPE                BYTE PARAMETER AUTOMATIC
                                              74   79

   12  0058H    28  REG                    BYTE ARRAY(28) DATA
                                             100  108  365  366  398  399

   12  002EH     2  REGINDEX               WORD
                                             107  108  145  380

   12  0074H    11  REGORD                 BYTE ARRAY(11) DATA
                                             230  242

   12  0030H    28  REGSAV                 WORD ARRAY(14)
                                             145  183  184  185  192  227  230  233  234
                                             238  239  242  245  254  261  262  263  301
                                             302  303  310  311  312  318  368  383  392
                                             404  414  430  511  522  590  591  597  631
                                             632  668  670  675  677

  236  0766H    96  RESTOREEXECUTE         PROCEDURE STACK=0008H
                                             256  264  304  313

  237  007FH    22  RESTOREEXECUTECODE     BYTE ARRAY(22) DATA
                                             237

  237  0004H     2  RESTOREEXECUTECODEPTR  WORD DATA
```

```
                                                246

  358  0054H     2  SAVE                        WORD
                                                389  392

  134  004CH     2  SAVE                        WORD
                                                139  145  149  154  158  159

  226  0715H    81  SAVEREGISTERS               PROCEDURE STACK=0002H
                                                253  275

   11  0002H     2  SEG                         WORD MEMBER(USERSTACK)
                                                233  238  246  250  272

   11  0002H     2  SEG                         WORD MEMBER(CSIP)
                                                184  262  289  302  311

   11  0002H     2  SEG                         WORD MEMBER(BRK1)
                                                668

   11  0002H     2  SEG                         WORD MEMBER(ARG3)
                                                540

  280  0002H     2  SEG                         WORD MEMBER(VECTOR)
                                                281

   11  0002H     2  SEG                         WORD MEMBER(ARG1)
                                                547  574  594

  169  0002H     2  SEG                         WORD MEMBER(ARG)
                                                170  175

                     SHL                        BUILTIN
                                                154  217  224

                     SHR                        BUILTIN
                                                 53   82  668

    8                SIO8251CMND                LITERALLY
                                                202  206  628

    8                SIO8251DTRON               LITERALLY
                                                196

    8                SIO8251MODE                LITERALLY
                                                626

    8                SIO8251RESET               LITERALLY
                                                620  624

   81  03D9H    14  SIO8251SETTLINGDELAY        PROCEDURE STACK=0002H
                                                621  623  625  627  629

    7  0023H     4  SIOBREAKMSG                 BYTE ARRAY(4) DATA
                                                671

   17  02A0H    19  SIOCHARRDY                  PROCEDURE BYTE STACK=0002H
```

```
                                        26   38

    22  02B3H    60  SIOCHECKCONTROLCHAR    PROCEDURE STACK=0006H
                                             39  209

     7  004EH    10  SIOCMND                BYTE ARRAY(10) DATA
                                             643  644

   113  04A7H    17  SIOCRLF                PROCEDURE STACK=0010H
                                             257  276  300  353  362  370  397  417  443
                                             468  492  530  534  543  549  561  566  580
                                             639

     8                SIODATAPORT            LITERALLY
                                             23   28   42   47

   427  0BE6H   192  SIODISPLAY              PROCEDURE STACK=0022H
                                             655

     8                SIODSRDY               LITERALLY
                                             198

   315  0926H   224  SIOEXAMMEM             PROCEDURE STACK=0022H
                                             650

   357  0A06H   335  SIOEXAMREG              PROCEDURE STACK=001CH
                                             651

   168  05E4H    65  SIOGETADDR             PROCEDURE STACK=001EH
                                             192  289  318  404  414  430  511  522

    44  030FH    33  SIOGETCHAR             PROCEDURE STACK=0010H
                                             119  123  126  129  131  142  146  155  165
                                             174  190  288  328  377  386  407  413  436
                                             472  483  495  502  507  510  514  521  642

   133  04F0H   244  SIOGETWORD             PROCEDURE WORD STACK=0014H
                                             171  176  333  389  408  437  465  480  484
                                             515  577

   284  0892H   109  SIOGO                  PROCEDURE STACK=0026H
                                             652

    92  0416H    27  SIOHEX                 PROCEDURE WORD STACK=0004H
                                             149  154  216  217

   462  0CA6H    64  SIOINPUT               PROCEDURE STACK=001CH
                                             656

   402  0B55H   145  SIOMOVE                 PROCEDURE STACK=0022H
                                             654

    62  0377H    11  SIOOUTBLANK            PROCEDURE STACK=0010H
                                             182  187  189  322  327  364  385  445  494

    51  0330H    48  SIOOUTBYTE             PROCEDURE STACK=0012H
                                             59   60   77   79  188  325  453  471  542
```

```
                                                548  557  560  565

     36  02EFH     32  SIOOUTCHAR                PROCEDURE STACK=000CH
                                                  49   53   54   63   69   75  114  115  186
                                                 326  365  366  367  382  384  398  399  493
                                                 532  568  610  611  640  641  662  674  676

     73  03B5H     36  SIOOUTHEADER              PROCEDURE STACK=0022H
                                                 539  546  555  564

    477  0CE6H     97  SIOOUTPUT                 PROCEDURE STACK=0018H
                                                 657

     65  0382H     51  SIOOUTSTRING              PROCEDURE STACK=0014H
                                                 630  671

     57  0360H     23  SIOOUTWORD                PROCEDURE STACK=0018H
                                                  78  183  324  354  368  383  444  448  470
                                                 540  541  547  675  677

    571  0ED8H    259  SIOREAD                   PROCEDURE STACK=001AH
                                                 658

    214  06CDH     43  SIOREADBYTE               PROCEDURE BYTE STACK=0010H
                                                 223  224  584  587  600  605

    194  0682H     75  SIOREADCHAR               PROCEDURE BYTE STACK=000AH
                                                 216  217  581

    221  06F8H     29  SIOREADWORD               PROCEDURE WORD STACK=0016H
                                                 585  590  591  594

      8              SIORXRDY                   LITERALLY
                                                  18   45  207

    128  04DEH     18  SIOSCANBLANK              PROCEDURE STACK=0014H
                                                 359  403  575

      7  0027H     23  SIOSIGNON                 BYTE ARRAY(23) DATA
                                                 630

    306  08FFH     39  SIOSINGLESTEP             PROCEDURE STACK=0026H
                                                 653

      8              SIOSTATPORT                LITERALLY
                                                  18   40   45  196  198  202  206  207  620
                                                 622  624  626  628

    117  04B8H     38  SIOTESTWORDMODE           PROCEDURE STACK=0014H
                                                 317  429  464  479

      8              SIOTXRDY                   LITERALLY
                                                  40

    181  0625H     93  SIOUPDATEIP               PROCEDURE STACK=0022H
                                                 258  285  307
```

```
 84  03E7H   47  SIOVALIDHEX          PROCEDURE BYTE STACK=0004H
                                       151  153

105  0460H   71  SIOVALIDREG          PROCEDURE BYTE STACK=0008H
                                       143  378

 98  0431H   47  SIOVALIDREGFIRST     PROCEDURE BYTE STACK=0002H
                                       140  374

500  0D47H  401  SIOWRITE             PROCEDURE STACK=0026H
                                       659

 12              SP                   LITERALLY
                                       239

 12              SS                   LITERALLY
                                       238  631

  6              SSCOMMAND            LITERALLY
                                       255

                 STACKBASE            BUILTIN
                                       250  252  272  274  638

                 STACKPTR             BUILTIN
                                       249  251  271  273  637

  6              STANDARDLEN          LITERALLY
                                       551  553

 13              STARTADDR            LITERALLY
                                        13   14

501  0072H    1  STARTREC             BYTE
                                       520  525  537

  6              STEPTRAP             LITERALLY
                                       254  263  303  312

 66  0000H    1  STR                  BYTE BASED(PTR) ARRAY(1)
                                        68   69

572  0077H    1  T                    BYTE
                                       600  601  605

428  0071H    1  T                    BYTE
                                       457  458

358  006FH    1  T                    BYTE
                                       376  378

222  006EH    1  T                    BYTE
                                       223  224

215  006DH    1  T                    BYTE
                                       216  217  218  219
```

```
    134  006CH     1  T                BYTE
                                         138  143  149  151

      6               TRUE             LITERALLY
                                          20   88  101  109  122  137  296  321  381
                                         503  520

      6               USERINITSP       LITERALLY
                                         632

     11  002AH     4  USERSTACK        STRUCTURE AT
                                         228  231  233  234  238  239  241  244  246
                                         249  250  271  272

    280  0000H     4  VECTOR           STRUCTURE BASED(INTVECTORPTR)
                                         281  282

    316  0052H     2  W                WORD
                                         333  338  339  343  344

    134  004EH     2  W                WORD
                                         136  158  159  161

     57  0004H     2  W                WORD PARAMETER AUTOMATIC
                                          58   59   60

      5  0069H     1  WORDMODE         BYTE
                                         118  122  323  336  350  446  458  469  487


MODULE INFORMATION:

     CODE AREA SIZE     = 0FDBH    4059D
     CONSTANT AREA SIZE = 0000H      0D
     VARIABLE AREA SIZE = 0078H     120D
     MAXIMUM STACK SIZE = 0042H      66D
     1211 LINES READ
     0 PROGRAM ERROR(S)

END OF PL/M-86 COMPILATION
```

INTEL CORPORATION, 3065 Bowers Avenue, Santa Clara, CA 95051 (408) 987-8080

Printed in U.S.A.

10267/993