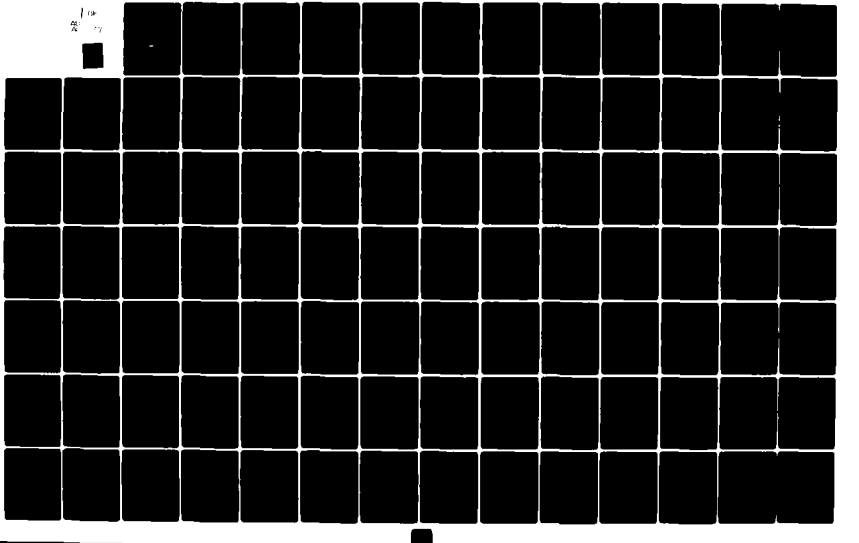
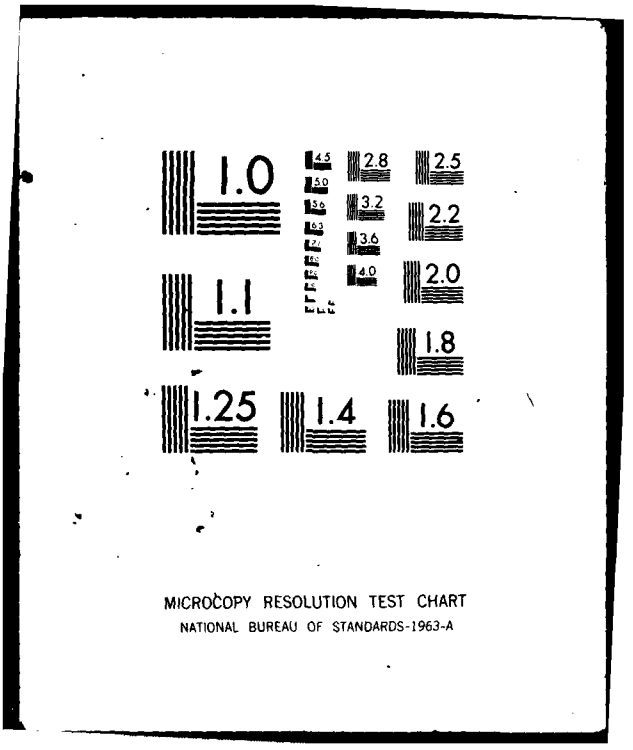
AD-A091 779 AIR FORCE ACADEMY CO F/G 9/2
USAFA/8086 - A STATE OF THE ART MICROPROCESSOR SYSTEM. VOLUME I--ETC(U)
JUN 80 J J POLLARD
UNCLASSIFIED USAFA-TR-80-16-VOL-2 NL




1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
6.3
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

LEVEL III

USAFA-TR-80-16 V. II

THE USAFA/8086

12 B.S.

A STATE OF THE ART MICROPROCESSOR SYSTEM

VOL. II. SOFTWARE DOCUMENTATION

CAPTAIN JOSEPH J. POLLARD

DEPARTMENT OF ELECTRICAL ENGINEERING

USAF ACADEMY, COLORADO 80840

JUNE 1980

FINAL REPORT

Technical Review by Major Robert W. Johnson
Department of Electrical Engineering
USAF Academy, Colorado 80840

Technical Review by Mr. Jim Eicher
AFFDL/FIGD
Wright-Patterson AFB, Ohio 45433

Editorial Review by Major Ahern
Department of English
USAF Academy, Colorado 80840

This research report is presented as a competent treatment of the subject, worthy of publication. The United States Air Force Academy vouches for the quality of the research, without necessarily endorsing the opinions and conclusions of the author.

This report has been cleared for open publication and/or public release by the appropriate Office of Information in accordance with AFR 190-17 and AFR 12-30. There is no objection to unlimited distribution of this report to the public at large, or by DDC to the National Technical Information Service.

This research report has been reviewed and is approved for publication.

M D Bacon

M. D. BACON, Colonel, USAF
Director of Research and
Continuing Education

DISCLAIMER NOTICE

THIS DOCUMENT IS BEST QUALITY PRACTICABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER: USAFA-TR-80-16 VOL-2	2. GOVT ACCESSION NO.: AD-A091779
	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle): USAFA/8086 - A State of the Art Microprocessor System. VOLUME II. Software Documentation. AD91 772	5. TYPE OF REPORT & PERIOD COVERED: Final Report
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): Captain Joseph J. Pollard	8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Department of Electrical Engineering, United States Air Force Academy, CO 80840	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 12 176
11. CONTROLLING OFFICE NAME AND ADDRESS: DFEE, USAF Academy, CO 80840	12. REPORT DATE: June 1980
	13. NUMBER OF PAGES: 172
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)	15. SECURITY CLASS. (of this report): Unclassified
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Distribution Unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Microprocessor, modules, USAF86, DOS86, DISK86, PIM86, ASM86 software

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report details the software developed at the U.S. Air Force Academy to support development of the USAFA/8086 microprocessor system. There are three primary modules consisting of the non-disk operating system (USAF86), the disk operating system (DOS86), and the disk interface software (DISK86). The first two modules are written in PIM86 while DISK86 is written in ASM86. An operator's manual is included as an appendix.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

011550

TABLE OF CONTENTS

APPENDIX

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Software was developed at the U.S. Air Force Academy to support development of the USAFA/8086 microprocessor system. There are three primary modules consisting of the non-disk operating system (USAFA86), the disk operating system (DOS86), and the disk interface software (DISK86). The first two modules are written in PLM86 while DISK86 is written in ASM86. An operator's manual is included as an appendix.

INTRODUCTION

The software for the USAFA/8086 was written at the U.S. Air Force Academy between August 1978 and April 1980. This software was written to the degree possible in a modular manner using PLM86, a high order language. The following personnel were responsible for the indicated modules with system software integration and debugging performed by Captain J. Pollard:

USAF86 Non Disk Operating System	Captain J. Pollard
DOS86 Disk Operating System	Lt P. Fitzjarrell Lt G. Rosenberger
DISK86 Disk Interface Routines	Lt D. McGrath

It is the purpose of this report to publish the software for future reference while at the same time providing drawings of the interrelations between procedures and short verbal descriptions of routine usage beyond the heavy comments provided in the actual code itself. The figures and diagrams of this report show only the interfaces between the various software routines and do not imply timing or sequencing information in any way. Figure 1 shows the intermodule relations.

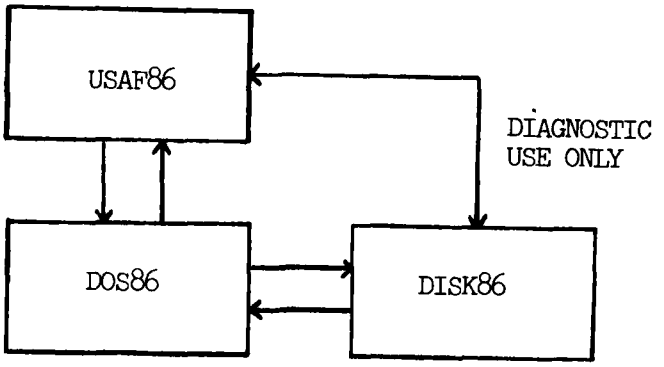


FIGURE 1. MODULE RELATIONSHIPS

I. USAF86 - THE NON-DISK OPERATING SYSTEM

The MON86 operating system was designed to support the various basic debugging functions considered desirable in a microprocessor system. Based upon the basic structure of the simple operating system supplied with the MCS-86 kits, this highly expanded and modified software module supports the following specific operating system commands:

S - Substitute
G - Execute (Go)
L - List to the Console Memory Data
P - Print to the Microprinter Memory Data
B - Set a single breakpoint
C - Clear the existing breakpoint
F - Fill a memory block
M - Move a memory block
Q - Convert a binary formatted floating point number to its ASCII representation and display the result
T - Test (Invoke the diagnostics sub-module)

In support of these commands within USAF86 there exist eight separate submodules supporting the main module USAF86:

1. Line Printer module
2. Console module
3. Arithmetic module
4. Console Switch/Led module
5. Diagnostic module
6. Command module
7. Interrupt Service module
8. Timer Module

Figure 2 shows the basic relations of USAF86 and its sub-modules.

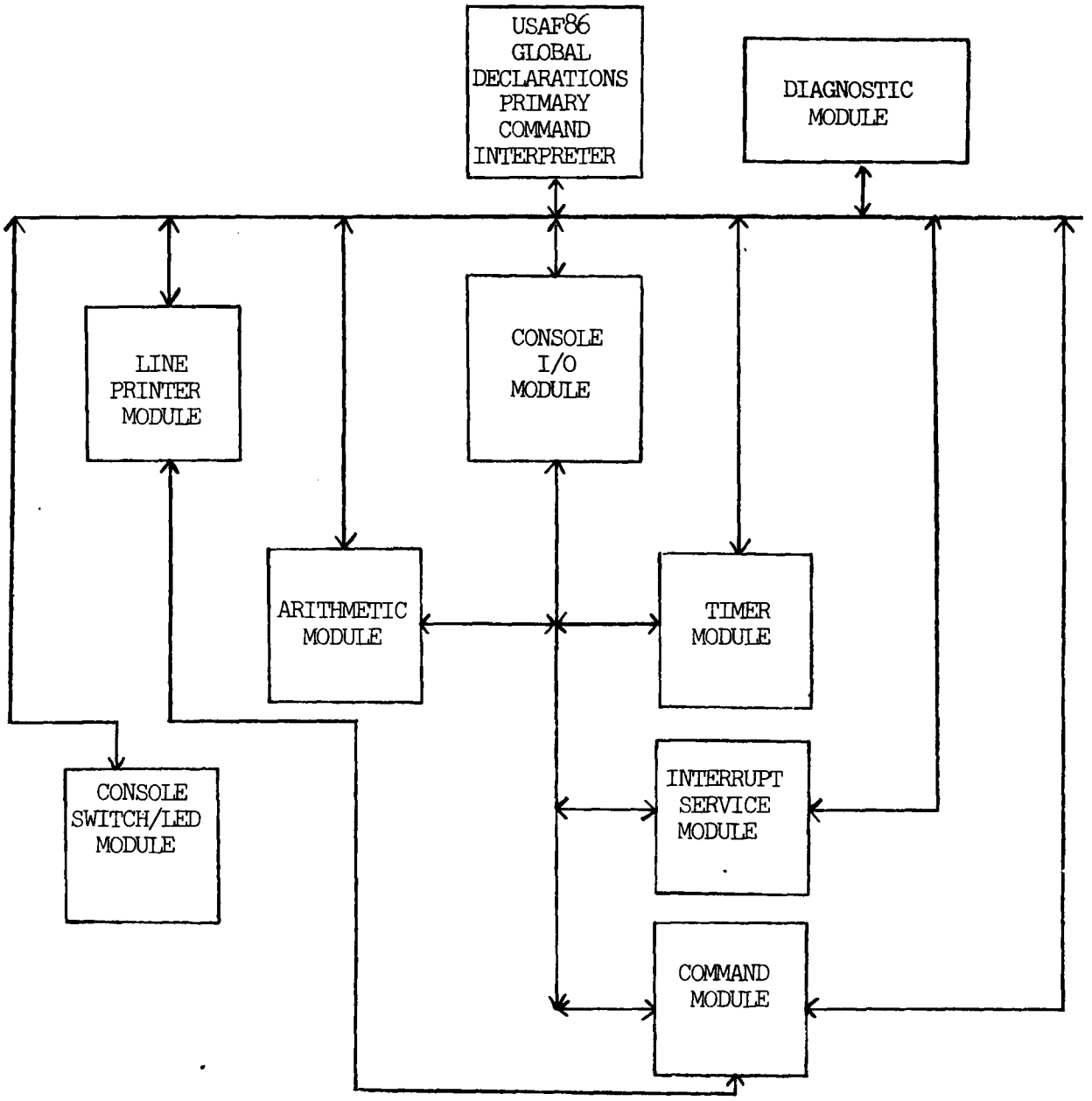


FIGURE 2. USAF86 SUBMODULE RELATIONS

Each of the submodules will now be diagrammed and discussed. The line printer module is shown in Figure 3. The module provides initialization capability (LP$INIT), plus the capability to output a single character (LPOUTCHAR), a byte (LPOUTBYTE), a word (LPOUTWORD), a string (LPOUTSTRING), blanks (LP$BLANKS), a carriage return and line feed (LP$CRLF), or a newline which consists of a carriage return, linefeed, and two blanks (LP$NEWLINE).

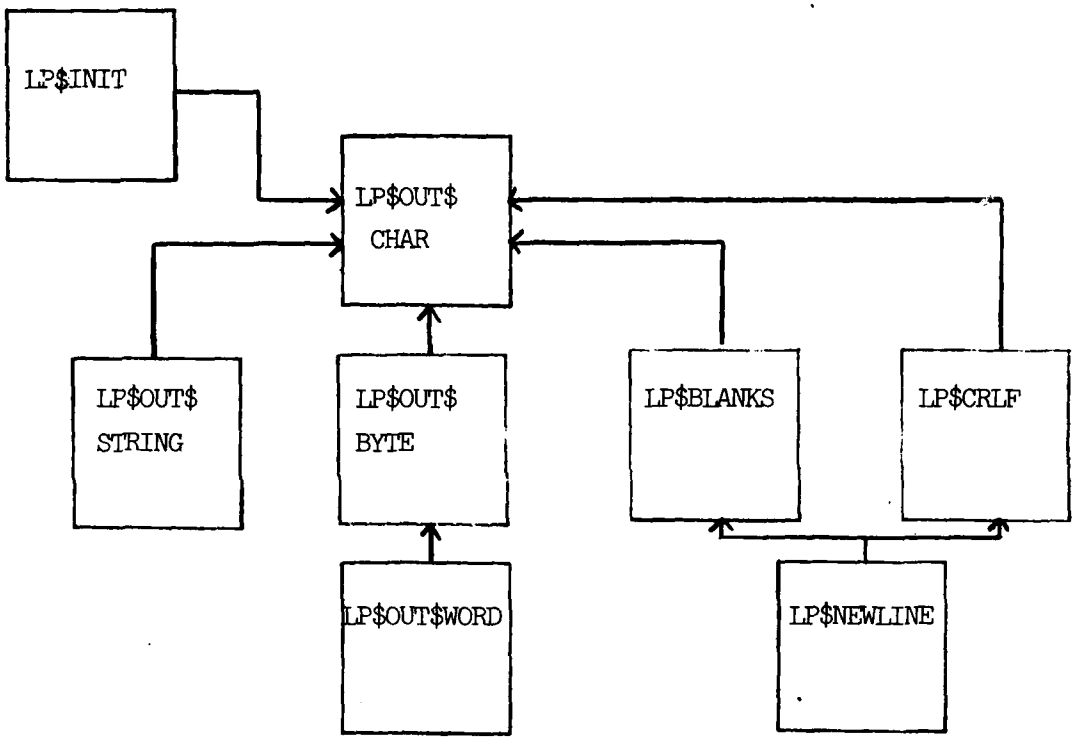


FIGURE 3. MICROPRINTER SUBMODULE RELATIONS

The console I/O module is the most complex of the submodules including some routines of a rather miscellaneous nature such as DELAY, DELAY$LONG, DISK$BUSY, INT$INIT, and SIO$INIT. Some of the routines could have been (and perhaps should have been) placed in other modules. The module basically provides the capability to retrieve and transmit data to either a serial or parallel console system. Figure 4 shows the relations of the various output procedures.

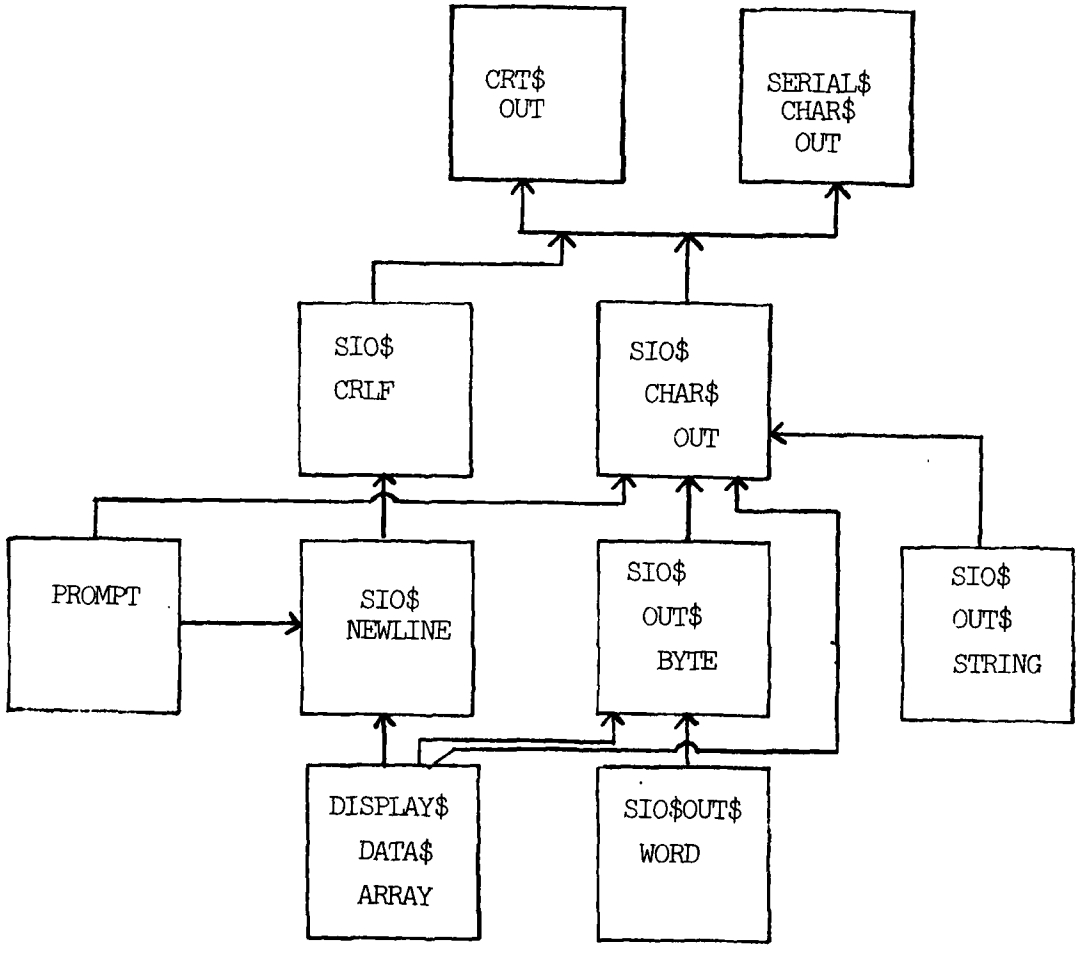


FIGURE 4. CONSOLE SUBMODULE OUTPUT PROCEDURES

Figure 5 shows the console submodule input procedures.

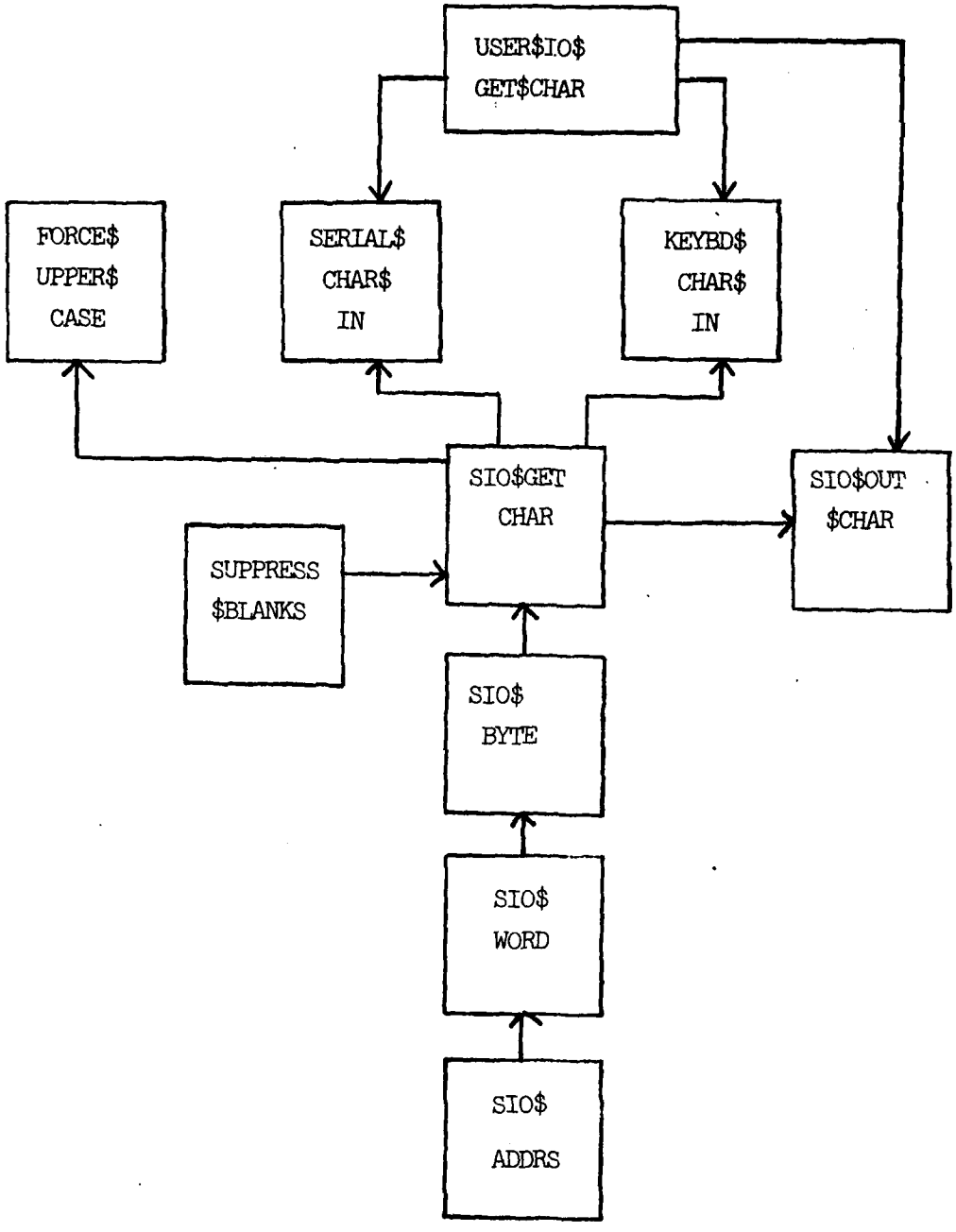


FIGURE 5. CONSOLE SUBMODULE INPUT PROCEDURES

In addition to these routines the module also contains **SIO$HEX** and **SIO$VALID$HEX** which create or determine the validity of hexadecimal characters.

The arithmetic module is relatively straight forward. This module provides a simple, though not necessarily optimal, method of communicating with the AM9511 Floating Point Processor board. Figure 6 shows the procedures.

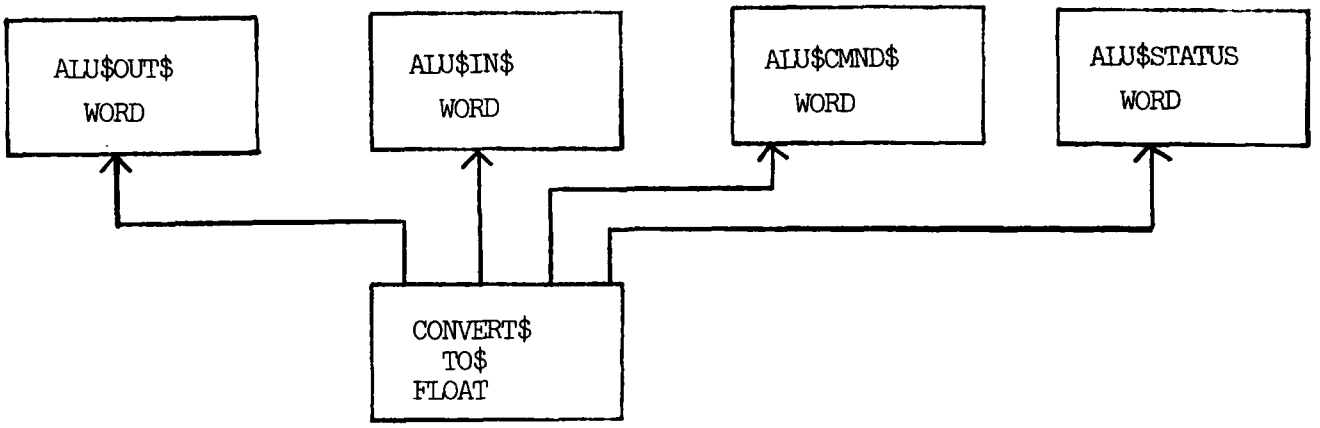


FIGURE 6. ARITHMETIC SUBMODULE

The console switches/leds submodule consists of only two independent routines READ$CONSOLE$SWITCHES and WRITE$CONSOLE$LEDS. The timer submodule is almost equally simple consisting of three independent routines used to initialize the timer (TIMER$INIT), load the timer (TIMER$LOAD), and read the timer (TIMER$READ).

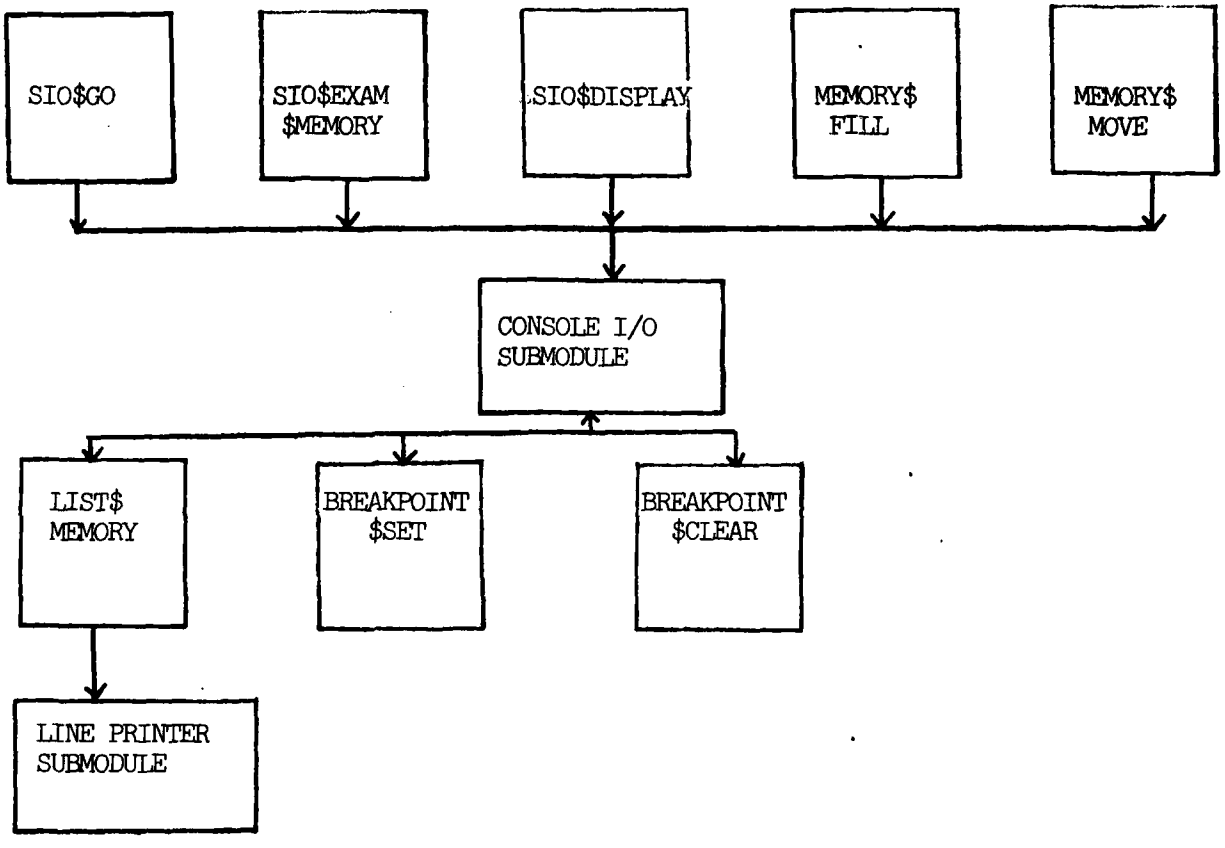


FIGURE 7. COMMAND SUBMODULE

The interrupt service submodule consists of eight procedures of a rather simplistic nature to artifically service the 8259 Interrupt Controller's requests plus four procedures to handle the system interrupts associated with the non-maskable interrupt, overflow errors, division by zero, and the software breakpoint. Additionally, a procedure is included for interrupt vector initialization.

The final submodule to be mentioned is the diagnostic module. This unit serves as a secondary control character interpreter allowing the user to test many of the features of the hardware. Essentially these procedures interact only through the console I/O module. The tests are best classified by function as shown in Table 1.

TABLE 1. SYSTEM DIAGNOSTICS

FUNCTION	CONTROL	TYPE
Ramp DACS	R	Hybrid
D/A driven by A/D	H	Hybrid
Hybrid Accuracy	L	Hybrid
Led/Switch Echo	C	Console
Keyboard/CRT	K	Console
Led Ramp	E	Console
LSI Circuits	I	System
Memory Test	M	System
APU Test	A	System
Discrete I/O	D	System
Timer	T	System
Floppy Disk	F	System

The basic operating system, USAF86, reserves certain memory locations by absolute allocation. These are shown in Table 2.

TABLE 2. ABSOLUTE MEMORY ALLOCATION

ABSOLUTE MEMORY LOCATION	FUNCTION
00000 to 00003	Divide by Zero Interrupt Vector
00008 to 0000B	NMI Vector
0000C to 0000F	Break Point Interrupt Vector
00010 to 00013	Overflow Interrupt Vector
00031 to 00034	Current Disk Track Storage
00036 to 00037	IOPB Address Storage
00080 to 0009F	Interrupt 20 to 27 Vectors
00100 to 00117	APU Scratch Pad Storage

The following pages contain the heavily commented listing of USAF86 plus the associated cross reference listing. To conserve paper no assembly code listing is provided.

```
ISIS-II PL/M-86 V1.1 COMPILATION OF MODULE USAF86
NO OBJECT MODULE REQUESTED
COMPILER INVOKED BY:  PLM86 :F1:CURRNT.SRC DATE(30 APR 80) NOOBJECT


           $TITLE('USAF-86') LARGE OPTIMIZE(2) XREF SYMBOLS INTVECTOR
  1        USAF86:DO;
               /* THIS SOFTWARE WAS WRITTEN AT THE
                  UNITED STATES AIR FORCE ACADEMY
                  DEPARTMENT OF ELECTRICAL ENGINEERING
                  (USAFA/DFEE) IN SUPPORT OF RESEARCH SPONSORED
                      BY
                  F. J. SEILER REREARCH LABORATORY  &
                  THE AIR FORCE FLIGHT DYNAMICS LABORATORY.
                  THE ORIGINAL STRUCTURE OF THIS PROGRAM WAS
                  MODELED ON THE INTEL CORPORATION, DEMO-86,
                  SOFTWARE SUPPLIED WITH THE PURCHASE OF THE
                  MCS-86 HARDWARE PACKAGE.  THIS OPERATING SYSTEM
                  IS INTENDED FOR USE ON THE USAFA/8086 MICROCOMPUTER
                  SYSTEM, WAS COMPILED, LINKED, AND LOCATED USING THE
                  INTEL MDS-311 SOFTWARE PACKAGE.  THIS SOFTWARE WAS
                  WRITTEN OVER THE PERIOD FROM AUG 1978 TO MAR 1980 BY
                  CAPT JOE POLLARD WITH SIGNIFICANT CONTRIBUTIONS FROM
                  MEMBERS OF THE CADET WING OF THE USAFA.
                THE PROGRAM IS MODULAR IN NATURE CONSISTING OF THE FOLLOWING
                  PRIMARY MODULES (SEE SYSTEM SOFTWARE DOCUMENTATION FOR
                  ADDITIONAL DETAILS):
                      1) MICROPRINTER  MODULE
                      2) AM9511 ARITHMETIC MODULE
                      3) CONSOLE I/O MODULE
                      4) COMMAND EXECUTION MOUULE
                      5) INTERRUPT SERVICE MODULE
                      6) COMMAND DISPATCH MODULE
                      7) DIAGNOSTICS MODULE
                      8) TIMER INTERFACE MODULE
                      9) DISK INTERFACE MODULE
                      10) HYBRID INTERFACE MODULE
                      11) CONSOLE SWITCH/LED INTERFACE MODULE
                      12) GLOBAL DATA DECLARATIONS        */
```

```
                $SUBTITLE('GLOBAL DATA DECLARATIONS')
                      /* THE BYTE CHAR IS USED AS A HOLDING QUEUE FOR CONSOLE
                         INPUT */
  2   1         DECLARE INT$VECTOR (100) POINTER AT (00000H);
  3   1         DECLARE
                      CHAR   BYTE PUBLIC;
  4   1         DECLARE
                      TRUE  LITERALLY '0FFH',
                   FALSE LITERALLY '000H';
  5   1         DECLARE
                   SIO$CMND(*) BYTE DATA ('GSLMFPBCQTRWKNERID');
                   /* THE AM9511 APU CONTROLLER USES I/O PORTS 10,12, AND 14 */
  6   1         DECLARE
                   ALU$AUTO$PORT LITERALLY '0014H',
                   ALU$CONTROL$PORT LITERALLY '0012H',
                   ALU$DATA$PORT LITERALLY '0010H';
                   /* THE SYSTEM USART IS LOCATED AT I/O PORTS 21 AND 23.  IT IS
                   CONFIGURED BY SOFTWARE FOR ASYNCHRONOUS, 8 BIT, NO PARITY, 16X
                   CLOCK TRANSMISSIONS.  BAUD RATE IS HARDWARE SELECTABLE.  TWO
                   STOP BITS ARE USED.  */
  7   1         DECLARE
                  SIO$STAT$PORT     LITERALLY '00023H',
                  SIO$DATA$PORT     LITERALLY '00021H'
                  SIO$8251$MODE   LITERALLY  '0CEH',
                  SIO$8251$RST      LITERALLY  '040H',
                  SIO$8251$CMND    LITERALLY  '025H',
                  SIO$RXRDY         LITERALLY  '02H',
                  SIO$TXRDY         LITERALLY  '01H';
                   /* THE SYSTEM INTERRUPT CONTROLLER (8259A) OCCUPIES I/O
                   PORTS 0 AND 2 OPERATING IN THE 8086 MODE AS A SINGLE
                   CONTROLLER.  */
  8   1         DECLARE
                   INT$MASK$PORT   LITERALLY '002H',
                   INT$STAT$PORT     LITERALLY '000H',
                   INT$ICW1        LITERALLY '013H',
                   INT$ICW2          LITERALLY '020H',
                   INT$ICW4    LITERALLY '00FH',
                   /* THE LINE PRINTER IS AT I/O PORT FC */
  9   1         DECLARE
                   LP$STATUS$PORT LITERALLY '00FCH',
                   LP$DATA$PORT   LITERALLY '00FCH';
 10   1         DECLARE
                   BREAK$POINT$DATA(2) BYTE;
 11   1         DECLARE
                   BREAK$POINT$FLAG BYTE;
 12   1         DECLARE
                   CONSOLE BYTE;
 13   1         DECLARE
                  ASCII(*)     BYTE DATA ('0123456789ABCDEF');
 14   1         DECLARE
                   ASTERISK       LITERALLY '2AH',
                   ASCR           LITERALLY '0DH',
                   ASLF           LITERALLY '0AH',
                   ASBL           LITERALLY '20H';
                   /* THE THREE FOLLOWING STRUCTURES ARE NECESSARY TO ALLOW THE
                   OPERATING SYSTEM TO HAVE INDIRECT ACCESS TO MEMORY */
```

```
15  1     DECLARE
             MEMORY$ARG1$PTR  POINTER PUBLIC,
             ARG1 STRUCTURE (OFF WORD, SEG WORD)
                AT (@MEMORY$ARG1$PTR),
             MEMORY$ARG1 BASED MEMORY$ARG1$PTR BYTE;
16  1     DECLARE MEMORY$ARG2$PTR POINTER,
             ARG2 STRUCTURE (OFF WORD, SEG WORD) AT (@MEMORY$ARG2$PTR),
             MEMORY$ARG2 BASED MEMORY$ARG2$PTR BYTE;
17  1     DECLARE BREAK$POINT$PTR POINTER,
             BKPTR STPUCTURE (OFF WORD, SEG WORD) AT (@BREAK$POINT$PTR),
             BREAK$POINT BASED BREAK$POINT$PTR BYTE;
             /* THE PARALLEL I/O CONSOLE IS AT I/O PORTS F0 AND F1  */
18  1     DECLARE
             KEYBD$STAT$PORT LITERALLY '00F1H',
             KEYBD$DATA$PORT LITERALLY '00F0H',
             CRT$PORT LITERALLY '00F0H',
             KEYBD$RDY LITERALLY '00H';
19  1     DECLARE
             CONSOLE$SWITCH$PORT LITERALLY '000FEH',
             CONSOLE$LED$PORT LITERALLY '000FEH';
             /* TIMER IS SET UP TO BE BINARY 16 BIT MODE 4 */
20  1     DECLARE
             TIMER$CONTROL$PORT LITERALLY '0037H',
             TIMER$BASE$PORT LITERALLY '0031H',
             TIMER0$MODE$WORD LITERALLY '034H',
             TIMER1$MODE$WORD LITERALLY '074H',
             TIMER2$MODE$WORD LITERALLY '0B4H';
21  1     DECLARE
             DISCRETE$IO$PORT1 LITERALLY '00F8H',
             DISCRETE$IO$PORT2 LITERALLY '00FAH';
22  1     DECLARE
             IOPB WORD PUBLIC AT (00036H); /* INPUT OUTPUT PARAMETER BLOCK FOR DISK IO */
23  1     DECLARE
             DERR BYTE, FTERR BYTE EXTERNAL; /* DISK ERROR WORDS */
24  1     DECLARE
             MAINPROGRAM LABEL PUBLIC;/* BOOTSTRAP ENTRY */
25  1     DECLARE
             (ERROR,NEXT$COMMAND) LABEL PUBLIC;
26  1     DISK:
             PROCEDURE EXTERNAL;
             /* THIS PROCEDURE IS THE DISK DRIVER PACKAGE WHICH COMMUNICATES
             WITH THE WESTERN DIGITAL 1791 DISK CONTROLLER.  THIS PROCEDURE
             IS WRITTEN IN ASM86 SINCE PARTS OF IT ARE TIME CRITICAL.  */
27  2          END;
28  1     DECLARE
             DISK$TRACK (4) BYTE PUBLIC AT (00031H);
29  1     DECLARE
             DISK$READY LITERALLY '80H',
             DISK$STATUS$PORT LITERALLY '00C8H',
             DISK$COMMAND$PORT LITERALLY '00C8H',
             DISK$RESET$PORT LITERALLY '00C4H',
             DISK$DRIVE$PORT LITERALLY '00C0H';
          /* THESE DEFINITIONS DEFINE THE I/O SPACE OF THE DISK CONTROLLER
             AND ARE USED WHILE INITIALIZING THE DISK SYSTEM */
30  1     READ$DRIVER:
             PROCEDURE EXTERNAL;
```

```
31  2       END;
32  1   WRITE$DRIVER:
            PROCEDURE EXTERNAL;
33  2       END;
34  1   COPY$DRIVER:
            PROCEDURE EXTERNAL;
35  2       END;
36  1   RENAME$DRIVER:
            PROCEDURE EXTERNAL;
37  2       END;
38  1   ERASE$DRIVER:
            PROCEDURE EXTERNAL;
39  2       END;
40  1   ATTRIBUTE$CHANGE$DRIVER:
            PROCEDURE EXTERNAL;
41  2       END;
42  1   INITIALIZE:
            PROCEDURE EXTERNAL;
43  2       END;
44  1   DIRECTORY$LIST:
            PROCEDURE EXTERNAL;
45  2       END;
```

```
                $SUBTITLE ('MICROPRINTER MODULE')
                   /* THIS MODULE IS DESIGNED FOR USE WITH A CENTRONICS MICROPRINTER
                      WITH PARALLEL INTERFACE  */
                   /* THE PRINTER IS INITIALIZED BY SELECTING 20 CPI AND NO UNDERLINE
                      MODE OF OPERATION */
   46   1      LP$INIT:
                   PROCEDURE;
   47   2         OUTPUT(LP$DATA$PORT)=160;
   48   2         OUTPUT(LP$DATA$PORT)=160;
   49   2         OUTPUT(LP$DATA$PORT)=350;
   50   2         END;
   51   1      LP$OUT$CHAR:
                   /* ONE ARGUMENT TYPE BYTE -- ASCII CHARACTER TO BE PRINTED */
                PROCEDURE(C) PUBLIC;
   52   2         DECLARE C BYTE;
                   /* THE LINE PRINTER READY BIT IS MONITORED ON BIT 1 OF THE STATUS
                      PORT.  */
   53   2      DO WHILE (INPUT(LP$STATUS$PORT) AND 02H)<>0;
   54   3         END;
                   /* THE CHARACTER TO BE XMITTED IS THE SEVEN LSBS WHILE THE
                   MSB IS THE STROBE WHICH IS SOFTWARE PULSED ACTIVE LOW */
   55   2         OUTPUT(LP$DATA$PORT)=C OR 80H;
   56   2         OUTPUT(LP$DATA$PORT)=C AND 7FH;
   57   2         OUTPUT(LP$DATA$PORT)=C OR 80H;
   58   2         END;
   59   1      LP$OUT$BYTE:
                   /* ONE ARGUMENT TYPE BYTE -- THE HEX BYTE WHICH IS TO BE
                   INTERPRETTED TO ASCII AND THEN PRINTED */
                   PROCEDURE(B) PUBLIC;
   60   2         DECLARE B BYTE;
   61   2         CALL LP$OUT$CHAR(ASCII(SHR(B,4) AND 0FH));
   62   2         CALL LP$OUT$CHAR(ASCII(B AND 0FH));
   63   2         END;
   64   1      LP$OUT$WORD:
                   /* ONE ARGUMENT TYPE WORD -- THE HEX WORD WHICH IS TO BE
                   INTERPRETTED TO ASCII AND THEN PRINTED */
                   PROCEDURE(W) PUBLIC;
   65   2         DECLARE W WORD;
   66   2         CALL LP$OUT$BYTE(HIGH(W));
   67   2         CALL LP$OUT$BYTE(LOW(W));
   68   2         END;
   69   1      LP$OUT$STRING:
                   /*TWO ARGUMENTS POINTER,BYTE -- THE POINTER TO THE ASCII STRING
                   TO BE PRINTED AND THE NUMBER OF BYTES IN THE STRING */
                   PROCEDURE(STR$PTR,N) PUBLIC;
   70   2         DECLARE STR$PTR POINTER, (N,I) BYTE,
                      STR BASED STR$PTR(1) BYTE;
   71   2         DO I=0 TO N-1;
   72   3         CALL LP$OUT$CHAR(STR(I));
   73   3         END;
   74   2         END;
   75   1      LP$BLANKS:
                   /* ONE ARGUMENT --THE NUMBER OF BLANKS TO BE PRINTED */
                   PROCEDURE (N) PUBLIC;
   76   2         DECLARE (I,N) BYTE;
   77   2         DO I=1 TO N;
```

```
 78  3          CALL LP$OUT$CHAR(ASBL);
 79  3          END;
 80  2          END;
 81  1      LP$CRLF:
            /* NO ARGUMENTS - AN ASCII CARRIAGE RETURN IS SENT TO THE LINE
            PRINTER.  A LINE FEED IS AUTOMATICALLY INSERTED BY THE HARDWARE  */
            PROCEDURE PUBLIC;
 82  2          CALL LP$OUT$CHAR(ASCR);
 83  2          END;
 84  1      LP$NEWLINE:
            /* NO ARGUMENTS - A CARRIAGE RETURN, AUTOMATIC LINE FEED, AND TWO
            BLANKS ARE SENT TO THE LINE PRINTER.  */
            PROCEDURE PUBLIC;
 85  2          CALL LP$CRLF;
 86  2          CALL LP$BLANKS(2);
 87  2          END;
```

```
                $SUBTITLE ('CONSOLE I/O')
 88   1         DELAY
                   /*ONE ARGUMENT  -- THE # OF ITERATIONS AT 4 MHZ  OPERATING IN
                      EPROM EACH ITERATION  TAKES APPROX 15 MICROSECONDS. */
                    PROCEDURE (C) PUBLIC;
 89   2            DECLARE C BYTE;
 90   2               DO WHILE C>0;
 91   3            C=C-1;
 92   3            END;
 93   2            END;
 94   1         DELAY$LONG:
                   /*NO ARGUMENTS --- A FIXED LENGTH DELAY ROUTINE USED WITH
                   CRT I/O ROUTINES. */
                   PROCEDURE PUBLIC;
 95   2            DECLARE I BYTE;
 96   2            DO I=1 TO  6;
 97   3            CALL TIME(5);
 98   3            END;
 99   2            END;
100   1         DISK$BUSY: PROCEDURE;/* THIS PROCEDURE WAITS FOR THE DISK TO BE NOT BUSY*/
101   2            CALL DELAY(20);
102   2            IF(((NOT(INPUT(DISK$STATUS$PORT))) AND DISK$READY)<>00H) THEN
103   2               RETURN;
104   2            DO WHILE (NOT(INPUT(DISK$STATUS$PORT)) AND 01H) <> 0;
105   3            END;
106   2            END DISK$BUSY;
107   1         INT$INIT:
                   /* SETS UP THE INTERRUPT CONTROLLER AT POWER ON. */
                   PROCEDURE;
108   2            OUTPUT(INT$STAT$PORT)=INT$ICW1;
                   /*  DELAYS ARE NECESSARY TO ALLOW LSI CIRCUIT RECOVERY */
109   2            CALL DELAY (20);
110   2            OUTPUT(INT$MASK$PORT)=INT$ICW2;
111   2            CALL DELAY(20);
112   2            OUTPUT(INT$MASK$PORT)=INT$ICW4;
113   2            END;
114   1         SIO$INIT:
                   /* SETS UP THE USART AT POWER ON. */
                   PROCEDURE;
115   2            OUTPUT(SIO$STAT$PORT)=080H;
116   2            CALL DELAY(20);
117   2            OUTPUT(SIO$STAT$PORT)=0;
118   2            CALL DELAY(2);
                   /* ISSUE AS SOFTWARE RESET */
119   2            OUTPUT(SIO$STAT$PORT)=SIO$8251$RST;
120   2            CALL DELAY(20);
                   /* SELECT THE DESIRED MODE OF OPERATION */
121   2            OUTPUT(SIO$STAT$PORT)=SIO$8251$MODE;
122   2            CALL DELAY(20);
123   2            OUTPUT(SIO$STAT$PORT)=SIO$8251$CMND;
124   2            END;
125   1         SERIAL$CHAR$OUT:
                   /* ONE ARGUMENT -- TYPE BYTE -- THE BYTE TO BE XMITTED BY THE
                   USART  THE XMITTER IS CHECKED AND IF READY THE CHARACTER IS
                   OUTPUT. */
                   PROCEDURE(C) PUBLIC;
```

```
126   2         DECLARE C BYTE;
127   2         DO WHILE(INPUT(SIO$STAT$PORT) AND SIO$TXRDY)=0
128   3         END;
129   2         OUTPUT(SIO$DATA$PORT)=C;
130   2         END;
131   1       CRT$OUT:
                /* ONE ARGUMENT- TYPE BYTE-- THE CHARACTER TO BE DISPLAYED ON THE CRT */   PROCEDURE(C) PUBLIC;
132   2         DECLARE C BYTE;
                /* OUTPUT THE CHARACTER WITH THE MSB (STROBE) LOW (ACTIVE) */
133   2         C=C AND 7FH;
134   2         OUTPUT(CRT$PORT)=C;
                /*DELAY 13 ITERATIONS FOR EVERY CHARACTER */
135   2         CALL DELAY(13);
                /* OUTPUT THE CHARACTER WITH THE STROBE HIGH */
136   2         OUTPUT(CRT$PORT)=C OR 80H;
                /* DELAY LONGER FOR SPECIAL CHARACTERS SEE VIDEO INTERFACE SPEC */
137   2         IF((C=0CH) OR (C=1CH)) THEN CALL TIME (6);
139   2         IF((C=1BH)OR(C=0AH)OR(C=1AH)OR(C=1DH)OR(C=0DH))THEN CALL DELAY$LONG;
                ELSE
141   2         CALL DELAY(13);
142   2         END;
143   1       SERIAL$CHAR$IN:
                /* READ A CHARACTER FROM THE USART WHEN READY */
                PROCEDURE PUBLIC;
144   2         DO WHILE(INPUT(SIO$STAT$PORT) AND SIO$RXRDY)=0;
145   3         END;
                /* STRIP THE PARITY BIT */
                /* RESULT IS RETURNED IN CHAR */
146   2         CHAR=INPUT(SIO$DATA$PORT) AND 7FH;
147   2         END;
148   1       KEYBD$CHAR$IN:
                /* READ CHARACTER FROM THE PARALLEL KEYBOARD WHEN AVAILABLE */
                PROCEDURE PUBLIC;
149   2         DO WHILE(INPUT(KEYBD$STAT$PORT) AND KEYBD$RDY)=0;
150   3         END;
                /*STRIP THE PARITY BIT */
                /* RETURN THE RESULT IN CHAR */
151   2         CHAR=INPUT(KEYBD$DATA$PORT) AND 7FH;
152   2         END;
153   1       FORCE$UPPER$CASE:
                /*IF A LOWER CASE CHARACTER IS DETECTED THEN FORCE UPPER CASE */
                PROCEDURE PUBLIC;
154   2         IF (CHAR>=61H) AND (CHAR <=7AH) THEN CHAR=CHAR -20H;
156   2         END;
157   1       SIO$OUT$CHAR:
                /* ONE ARGUMENT -- TYPE BYTE -- ASCII CHARACTER TO BE
                DISPLAYED AT THE CONSOLE  */
                PROCEDURE (C) PUBLIC;
158   2         DECLARE C BYTE;
159   2         IF CONSOLE=1 THEN CALL SERIAL$CHAR$OUT(C);
161   2         IF CONSOLE=2 THEN CALL CRT$OUT(C);
163   2         END;
164   1       SIO$OUT$BYTE:
                /* ONE ARGUMENT -- TYPE BYTE -- HEX BYTE TO BE DISPLAYED ON
                THE CONSOLE  */
                PROCEDURE (B) PUBLIC;
```

```
165  2          /* DISPLAY A BYTE
166  2          CALL SIO$OUT$CHAR(ASCII(SHR(B,4) AND 0FH));
167  2          CALL SIO$OUT$CHAR(ASCII(B AND 0FH));
168  2          END;
169  1      SIO$OUT$WORD:
               /* ONE ARGUMENT--TYPE WORD--HEX WORD TO BE DISPLAYED ON THE CONSOLE */
               PROCEDURE (W) PUBLIC;
170  2          DECLARE W WORD;
171  2          CALL SIO$OUT$BYTE(HIGH(W));
172  2          CALL SIO$OUT$BYTE(LOW(W));
173  2          END;
174  1      SIO$OUT$STRING:
               /* TWO ARGUMENTS--POINTER,BYTE--THE POINTER TO THE STRING TO BE
               DISPLAYED ON THE CONSOLE AND THE # OF CHARACTERS TO BE DISPLAYED. */
               PROCEDURE (STR$PTR,N) PUBLIC;
175  2          DECLARE STR$PTR POINTER,(N,I) BYTE,
                   STR BASED STR$PTR (1) BYTE;
176  2          DO I=0 TO N-1;
177  3          CALL SIO$OUT$CHAR(STR(I));
178  3          END;
179  2          END;
180  1      SIO$GET$CHAR:
               /* READ A CHARACTER FROM THE CURRENT CONSOLE. */
               PROCEDURE PUBLIC;
181  2          IF CONSOLE=1 THEN CALL SERIAL$CHAR$IN;
183  2          IF CONSOLE=2 THEN CALL KEYBD$CHAR$IN;
185  2          CALL FORCE$UPPER$CASE;
               /* ECHO THE CHARACTER TO THE CONSOLE */
186  2          CALL SIO$OUT$CHAR(CHAR);
               /* IF THE CHARACTER WAS AN X ABORT THE COMMAND IN PROGRESS */
187  2          IF CHAR='X' THEN GO TO NEXT$COMMAND;
189  2          END;
190  1      SIO$VALID$HEX:
               /* THIS FUNCTIONAL ROUTINE VERIFIES A BYTE AS A VALID ASCII
               REPRESENTATION OF A HEX CHARACTER (0-9,A-F). */
                  PROCEDURE (H) BYTE PUBLIC;
191  2          DECLARE H BYTE;
192  2          IF('0'<=H AND H<='9') OR ('A'<=H AND H<='F')  THEN
193  2          RETURN TRUE;
               ELSE
194  2          RETURN FALSE;
195  2          END;
196  1      SIO$HEX:
               /* THIS FUNCTIONAL ROUTINE RETURNS A HEX NUMBER GIVEN A VALID
               ASCII REPRESENTATION. */
                PROCEDURE (C) BYTE PUBLIC;
197  2          DECLARE C BYTE;
198  2          C = C - 30H;
199  2          IF C <=9 THEN RETURN C;
201  2          ELSE RETURN C - 7;
202  2          END;
203  1      SIO$GET$BYTE:
               /* THIS ROUTINE MUST HAVE A VALID CHARACTER QUEUED IN CHAR ON
               ENTRY.  THIS ROUTINE OBTAINS A VALID HEX BYTE FROM THE CONSOLE
               IF POSSIBLE. */
               PROCEDURE BYTE PUBLIC;
```

```
204a  2         DECLARE B BYTE
                /* ABORT ON INVALID CHARACTERS */
205   2         IF NOT SIO$VALID$HEX(CHAR) THEN GO TO ERROR;
207   2         B = 0;
                /* PACK THE NEXT BYTE. ONLY ONE BYTE IS RETURNED
                CONSISTING OF THE LAST TWO VALID ENTRIES */
208   2         DO WHILE SIO$VALID$HEX(CHAR);
209   3         B = SHL(B,4)+SIO$HEX(CHAR);
210   3         CALL SIO$GET$CHAR;
211   3         END;
                /* EXIT CONDITIONS ARE CR,BLANK, OR COMMA. */
212   2         IF CHAR = ASCR OR CHAR = ASBL OR CHAR = ',' THEN RETURN B;
214   2         GO TO ERROR;
215   2         END;
216   1     SIO$GET$WORD:
                /* ROUTINE GETS A WORD FROM THE CONSOLE. ADDITION AND SUBTRACTION
                ARE PERMITTED OF A SECOND WORD PRIOR TO ENTRY. */
                PROCEDURE WORD PUBLIC;
217   2         DECLARE (SAVE,W) WORD, OPER BYTE;
218   2         OPER = '+';
219   2         W = 0;
220   2         SAVE = 0;
                /* FORM A WORD ALWAYS PLACING THE CURRENT WORD IN SAVE */
221   2         DO WHILE TRUE;
222   3         DO WHILE SIO$VALID$HEX(CHAR);
223   4         SAVE = SHL(SAVE,4) + DOUBLE(SIO$HEX(CHAR));
224   4         CALL SIO$GET$CHAR;
225   4         END;
                /* IF ADD OR INITIAL INPUT STORE IN W */
226   3         IF OPER='+' THEN
227   3         W = W + SAVE;
                ELSE
                /*OTHERWISE SUBTRACT */
228   3         W = W - SAVE;
                /* TERMINATES ON COLON,COMMA, OR BLANK */
229   3         IF CHAR=ASCR OR CHAR=':' OR CHAR=',' OR CHAR=ASBL THEN RETURN W;
231   3         IF CHAR='+' OR CHAR='-' THEN
232   3         DO;
233   4         OPER = CHAR;
234   4         SAVE=0;
235   4         END;
                ELSE
                /* ABORTS IF ENTRY IS NOT VALID THROUGH SYSTEM ERROR ROUTINE */
236   3         GO TO ERROR;
237   3         CALL SIO$GET$CHAR;
238   3         END;
239   2         END;
240   1     SIO$GET$ADDR:
                /* ROUTINE GETS AN ADDRESS (SEGMENT:OFFSET) ONE ARGUMENT--TYPE WORD--
                A DEFAULT SEGMENT. */
                 PROCEDURE(DEFAULT$BASE) PUBLIC;
241   2         DECLARE DEFAULT$BASE WORD;
242   2         ARG1.SEG = DEFAULT$BASE;
243   2         ARG1.OFF = SIO$GET$WORD;
244   2         DO WHILE CHAR = ':';
245   3         CALL SIO$GET$CHAR;
```

```
246  3          ARG1 SET = ARG1 OFF;
247  2          ARG1 OFF = SIO$GET$WORD;
248  3          END;
249  2          END;
250  1      SIO$BLANKS:
               /* OUTPUTS N BLANKS TO THE CONSOLE */
                PROCEDURE(N) PUBLIC;
251  2          DECLARE (I,N) BYTE;
252  2          DO I=1 TO N;
253  3          CALL SIO$OUT$CHAR(ASBL);
254  3          END;
255  2          END;
256  1      SIO$CRLF:
               /* OUTPUTS A CARRIAGE RETURN, LINEFEED TO THE CONSOLE */
                PROCEDURE PUBLIC;
257  2          CALL SIO$OUT$CHAR(ASCR);
258  2          IF CONSOLE=1 THEN CALL SIO$OUT$CHAR(ASLF);
260  2          IF CONSOLE=2 THEN CALL CRT$OUT(1BH);
262  2          IF CONSOLE=2 THEN CALL CRT$OUT(1AH);
264  2          END;
265  1      SIO$NEWLINE:
               /* OUTPUTS A CARRIAGE RETURN, LINE FEED TO THE
               CONSOLE. */
                PROCEDURE PUBLIC;
266  2          CALL SIO$CRLF;
267  2          END;
268  1      PROMPT:
               /* OUTPUTS A LINEFEED, CR, AND ASTERISK TO THE CONSOLE */
               PROCEDURE;
269  2          CALL SIO$NEWLINE;
270  2          CALL SIO$OUT$CHAR(ASTERISK);
271  2          END;
272  1      DISPLAY$DATA$ARRAY:
               PROCEDURE(A,N) PUBLIC;
                   /* THIS PROCEDURE DISPLAYS N DATA BYTES IN HEX BEGINNING
                   AT POINTER A.  EVERY 16 BYTES STARTS A NEW LINE. */
273  2          DECLARE A POINTER, N BYTE, I BYTE, M BYTE;
274  2          DECLARE B BASED A (1) BYTE;
275  2          DO WHILE TRUE;
276  3          IF (N>16) THEN M=16;
               ELSE
278  3          M=N;
279  3          N=N-M;
280  3          DO I=0 TO M-1;
281  4          CALL SIO$OUT$CHAR(' ');
282  4          CALL SIO$OUT$BYTE(B(I));
283  4          CALL SIO$OUT$CHAR(' ');
284  4          END;
285  3          CALL SIO$NEWLINE;
286  3          IF M<16 THEN RETURN;
288  3          END;
289  2          END;
290  1      CONSOLE$IDENT
               /* IDENTIFIES THE CONSOLE AS EITHER A SERIAL DEVICE (1) OR A
               PARALLEL DEVICE (2) DURING SYSTEM INITIALIZATION.  FIRST
               DEVICE TO PRESENT A CHARACTER IS SELECTED AS THE CONSOLE. */
```

```
                    PROCEDURE
291   2             CONSOLE=0;
292   2             DO WHILE CONSOLE=0;
293   3             IF(INPUT(SIO$STAT$PORT) AND SIO$RXRDY)<>0 THEN CONSOLE=1;
295   3             IF(INPUT(KEYBD$STAT$PORT) AND KEYBD$RDY)<>0 THEN CONSOLE=2;
297   3             END;
                    /* STRIP PARITY, STORE RESULT IN CHAR */
298   2             IF CONSOLE =1 THEN CHAR=INPUT(SIO$DATA$PORT) AND 7FH;
300   2             IF CONSOLE =2 THEN CHAR =INPUT(KEYBD$DATA$PORT) AND 7FH;
302   2             CALL SIO$OUT$CHAR(CHAR);
                    /* NOTE: THE TTY     WILL ALWAYS APPEAR READY IF USART IS NOT IN THE
                    SYSTEM.  SIMILARLY THE KEYBOARD LOOKS READY IF KEYBD STATUS PORT
                    IS REMOVED.  */
302   2             END;
304   1         SUPPRESS$BLANKS:
                    PROCEDURE PUBLIC;
                       /* THIS PROCEDURE GETS ONE CHARACTER FROM THE CONSOLE
                       SUPPRESSING ALL LEADING BLANKS.  */
305   2             CALL SIO$GET$CHAR;
306   2             DO WHILE CHAR=ASBL;
307   3             CALL SIO$GET$CHAR;
308   3             END;
309   2             END;
310   1         USER$IO$GET$CHAR:
                    PROCEDURE PUBLIC;
                       /* THIS PROCEDURE IS IDENTICAL TO SIO$GET$CHAR BUT
                       DOES NOT ABORT AUTOMATICALLY ON THE CHARACTER X.  */
311   2             IF CONSOLE=1 THEN CALL SERIAL$CHAR$IN;
313   2             IF CONSOLE=2 THEN CALL KEYBD$CHAR$IN;
315   2             CALL FORCE$UPPER$CASE;
316   2             CALL SIO$OUT$CHAR(CHAR);
317   2             END;
```

```
$SUBTITLE ('ARITHMETIC MODULE')
    /* APU WORDS ARE STORED EXPONENT IN LOWEST ADDRESSED BYTE */
    /* APU FLOATING POINT WORDS MUST START IN A LOCATION WHERE
    THE TWO LSB'S ARE ZERO I.E., 0,4,8 OR C AS THE LAST DIGIT. */
318  1  ALU$OUT$WORD:
        /* GIVEN AN ARBITRARY ADDRESS (SEG:OFF) FIND AN ADDRESS USED BY THE APU
        AND OUTPUT 4 BYTES OF DATA AUTOMATICALLY TO THE APU. */
        PROCEDURE(FLT$ADR) PUBLIC;
319  2      DECLARE FLT$ADR POINTER;
320  2      DECLARE FLT$PTR POINTER,
            FLT$ADRS STRUCTURE(OFF WORD, SEG WORD) AT (@FLT$PTR);
321  2      FLT$PTR=FLT$ADR;
            /* TRUNCATE THE 4 MSB AND ISSUE ADDRESS TO CONTROLLER FOR
            AUTO XFER TO AM9511 */
322  2      FLT$ADRS.OFF=FLT$ADRS.OFF+SHL(FLT$ADRS.SEG,4);
323  2      OUTWORD(ALU$AUTO$PORT)=FLT$ADRS.OFF AND 0FFFCH;
324  2      END;
325  1  ALU$IN$WORD:
        /* SAME AS ABOVE EXCEPT THE XFER IS FROM THE APU INTO MEMORY */
        PROCEDURE(FLT$ADR) PUBLIC;
326  2      DECLARE FLT$ADR POINTER;
327  2      DECLARE FLT$PTR POINTER,
            FLT$ADRS STRUCTURE(OFF WORD, SEG WORD)  AT (@FLT$PTR);
328  2      FLT$PTR=FLT$ADR;
329  2      FLT$ADRS.OFF=FLT$ADRS.OFF+SHL(FLT$ADRS.SEG,4);
330  2      OUTWORD(ALU$AUTO$PORT)=(FLT$ADRS.OFF AND 0FFFCH) OR 0001H;
331  2      END;
332  1  ALU$CMND$WORD:
        /* ONE ARGUMENT--AN AM9511 INSTRUCTION CODE */
        PROCEDURE (B) PUBLIC;
333  2      DECLARE B BYTE;
334  2      OUTPUT(ALU$CONTROL$PORT)=B;
335  2      END;
336  1  ALU$STATUS$WORD:
        /* FUNCTIONAL ROUTINE WHICH RETURNS THE AM9511 STATUS WORD */
        PROCEDURE BYTE PUBLIC;
337  2      RETURN INPUT(ALU$CONTROL$PORT);
338  2      END;
339  1  CONVERT$TO$FLOAT:
        /* OPERATING SYSTEM ROUTINE USED TO DISPLAY FLOATING POINT NUMBERS
        ON THE CONSOLE  */
        PROCEDURE (N) PUBLIC;/* ONE ARGUMENT TYYPE BYTE IF 0 THIS ROUTINE
                DISPLAYS THE FLOATING POINT NUMBER POINTED TO
                BY MEM$ARG1$PTR. */
340  2      DECLARE N BYTE;
341  2      DECLARE FLT$ADRS POINTER,FLT$ADR STRUCTURE(OFF WORD,SEG WORD) AT (@FLT$ADRS),
            FLT$DATA BASED FLT$ADRS (4) BYTE;
            /* USES MEMORY AT 00100 AS SCRATCH PAD  */
342  2      DECLARE SCRATCH(4) BYTE AT (00100H);
343  2      DECLARE ONE(4) BYTE AT (00104H);
344  2      DECLARE TEN(4) BYTE AT (00108H);
345  2      DECLARE ONES(4) BYTE DATA (00H,0FFH,0FFH,0FEH);
346  2      DECLARE ALMOST$TEN(4) BYTE AT (00114H);
347  2      DECLARE ALMOST$TENS(4) BYTE DATA (04H, 9FH,0FFH,0FFH);
348  2      DECLARE TENS(4) BYTE DATA (04H,0A0H,00H,00H);
349  2      DECLARE RES(4) BYTE AT (0010CH);
```

```
350   2          DECLARE COUNT BYTE
351   2             DECLARE DISPLAY$CHAR(4) BYTE AT (00110H);
352   2          DECLARE SAVE$DISPLAY(5) BYTE;
353   2          DECLARE (E,FLEAF2,EXP$SIGN,MSIGN, I) BYTE;
                     /*GET A CHARACTER */
354   2          N=N+1-1; /*DUMMY FOR COMPILER */
355   2          IF N=0 THEN GO TO FLOATING$DISPLAY;
357   2          CALL SUPPRESS$BLANKS;
                 /* GET AN ADDRESS; DEFAULT SEG IS ZERO */
358   2          CALL SIO$GET$ADDR(0);
359   2       FLOATING$DISPLAY:
                 COUNT=00H;
360   2          FLT$ADR OFF=ARG1 OFF AND 0FFFCH;
361   2          FLT$ADR SEG=ARG1 SEG;
362   2          IF (FLT$DATA(0)=0) AND (FLT$DATA(1)=0) AND (FLT$DATA(2)=0) AND
                     (FLT$DATA(3)=0) THEN GO TO ZERO$EXIT;
                 /* SET UP SCRATCH AREA  */
364   2          DO I=0 TO 3;
365   3          SCRATCH(I)=FLT$DATA(I);
366   3          ONE(I)=ONES(I);
367   3          ALMOST$TEN(I)=ALMOST$TENS(I);
368   3          TEN(I)=TENS(I);
369   3          END; •
                 /* STRIP THE MANTISSA SIGN */
370   2          MSIGN=SCRATCH(0) AND 80H;
371   2          SCRATCH(0) = SCRATCH(0) AND 7FH;
                 /* CHECK FOR A NUMBER GREATER OR EQUAL TO ONE AND LESS THAN
                 TEN.  IF GREATER THAN TEN DIVIDE REPEATEDLY BY TEN INCREASING
                 THE EXPONENT APPROPRIATELY.  IF LESS THAN ONE MULTIPLY BY TEN;
                 DECREASING THE EXPONENT EACH TIME */
372   2       TST$LP1:CALL ALU$OUT$WORD(@ALMOST$TEN(0));
373   2          CALL ALU$OUT$WORD(@SCRATCH(0));
374   2          CALL ALU$CMND$WORD(11H); /*9.99-SCRATCH */
375   2          CALL ALU$IN$WORD(@RES(0));
376   2          IF (RES(0) AND 80H) <> 0 THEN GO TO DIVIDE$LOOP;
378   2       TST$LP2:CALL ALU$OUT$WORD(@ONE(0));
379   2          CALL ALU$OUT$WORD(@SCRATCH(0));
380   2          CALL ALU$CMND$WORD(11H);
381   2          CALL ALU$IN$WORD(@RES(0));
382   2          IF(RES(0) AND 80H) = 0 THEN GO TO MULT$LOOP;
384   2          GO TO PROCESS;
385   2       MULT$LOOP:
                 EXP$SIGN=1;
386   2          COUNT=COUNT+1;
387   2          CALL ALU$OUT$WORD(@SCRATCH(0));
388   2          CALL ALU$OUT$WORD(@TEN(0));
389   2          CALL ALU$CMND$WORD(12H);
390   2          CALL ALU$IN$WORD(@SCRATCH(0));
391   2          GO TO TST$LP2;
392   2       DIVIDE$LOOP:
                 EXP$SIGN=0;
393   2          COUNT=COUNT+1;
394   2          CALL ALU$OUT$WORD(@SCRATCH(0));
395   2          CALL ALU$OUT$WORD(@TEN(0));
396   2          CALL ALU$CMND$WORD(13H);
397   2          CALL ALU$IN$WORD(@SCRATCH(0));
```

```
398  2          GO TO TEST$LP;
                /* STRIP THE MANTISSA DIGITS OFF BY MULTIPLYING AND INTEGERIZING */
399  2       PROCESS:
                CALL ALU$OUT$WORD(@SCRATCH(0));
400  2          DO I=0 TO 7;
401  3          IF I=0 THEN GO TO BY$PASS$MULTIPLY;
402  3          CALL ALU$OUT$WORD(@TEN(0));
403  3          CALL ALU$CMND$WORD(12H); /* MULTIPLY BY TEN */
404  3       BY$PASS$MULTIPLY:
                CALL ALU$CMND$WORD(17H); /* PUSH A COPY */
405  3          CALL ALU$CMND$WORD(1EH); /* FIX IT */
406  3          CALL ALU$CMND$WORD(37H); /* PUSH A COPY */
                /* SAVE IN AN ARRAY */
407  3          CALL ALU$IN$WORD (@DISPLAY$CHAR);
408  3          SAVE$DISPLAY(I)=DISPLAY$CHAR(3);
409  3          CALL ALU$CMND$WORD(1CH); /* FLOAT THE FIXED COPY */
410  3          CALL ALU$CMND$WORD(11H); /* SUBTRACT */
411  3          END;
412  2          CALL SIO$BLANKS(5);
                /* OUTPUT THE SIGN OF THE MANTISSA  */
413  2          IF MSIGN<> 0 THEN CALL SIO$OUT$CHAR('-');
                ELSE
414  2          CALL SIO$OUT$CHAR('+');
                /* OUTPUT THE MOST SIGNIFICANT DIGIT */
415  2          CALL SIO$OUT$CHAR(ASCII(SAVE$DISPLAY(0)));
                /* OUTPUT THE DECIMAL POINT */
416  2          CALL SIO$OUT$CHAR('.');
                /* OUTPUT THE REMAINDER OF THE MANTISSA */
417  2          DO I=1 TO 7;
418  3          CALL SIO$OUT$CHAR(ASCII(SAVE$DISPLAY(I)));
419  3          END;
                /* FIND THE EXPONENT */
420  2          EXP1= COUNT MOD 10;
421  2          COUNT=COUNT/10;
422  2          EXP2= COUNT MOD 10;
                /* OUTPUT A BLANK */
423  2          CALL SIO$OUT$CHAR (' ');
                /* OUTPUT AN E FOR EXPONENT */
424  2          CALL SIO$OUT$CHAR ('E');
                /* OUTPUT THE EXPONENT SIGN */
425  2          IF EXP$SIGN<>0 THEN CALL SIO$OUT$CHAR('-');
                ELSE
427  2          CALL SIO$OUT$CHAR('+');
                /* OUTPUT THE EXPONENT */
428  2          CALL SIO$OUT$CHAR(ASCII(EXP2));
429  2          CALL SIO$OUT$CHAR(ASCII(EXP1));
430  2          RETURN;
431  2       ZERO$EXIT:
                CALL SIO$BLANKS(5);
432  2          CALL SIO$OUT$STRING(@('0.00000 E+00'),12);
433  2          END;
            $INCLUDE (:F1:CONSOL.PLM)
```

```
        =   $SUBTITLE (CONSOLE SWITCH/LED MODULE)
        =           /* THESE ROUTINES ALLOW THE CONSOLE SWITCHES TO
        =           BE READ AND THE CONSOLE LEDS TO BE WRITTEN USING SYSTEM CALLS */
436   1 =   READ$CONSOLE$SWITCHES:
        =       PROCEDURE WORD PUBLIC;
437   2 =       RETURN INWORD(CONSOLE$SWITCH$PORT);
438   2 =       END;
439   1 =   WRITE$CONSOLE$LEDS:
        =       PROCEDURE(VALUE) PUBLIC;
440   2 =       DECLARE VALUE WORD;
441   2 =       OUTWORD(CONSOLE$LED$PORT)=VALUE;
442   2 =       END;
            $INCLUDE (:F1:TIMER.PLM)
```

```
        =   $SUBTITLE( TIMER MODULE )
        =       /* THIS MODULE IS DEVELOPED FOR THE 8253 TIMER USED IN THE
        =          USER'S 8086 SYSTEM.  GLOBAL DECLARATIONS OF THE
        =          TIMER MODE WORDS AND TIMER PORTS ARE REQUIRED
        =          FOR COMPILATION. */
443   1 =   TIMER$INIT
        =          PROCEDURE;
444   2 =          OUTPUT(TIMER$CONTROL$PORT)=TIMER0$MODE$WORD;
445   2 =          CALL DELAY(2);
446   2 =          OUTPUT(TIMER$CONTROL$PORT)=TIMER1$MODE$WORD;
447   2 =          CALL DELAY(2);
448   2 =          OUTPUT(TIMER$CONTROL$PORT)=TIMER2$MODE$WORD;
449   2 =          CALL DELAY(2);
450   2 =          END;
451   1 =   TIMER$LOAD:
        =          PROCEDURE(VALUE,N) PUBLIC;
452   2 =          DECLARE VALUE WORD,N WORD,DATA$PORT WORD;
453   2 =          IF N>2 THEN GO TO TIMER$ERROR;
455   2 =          DATA$PORT=TIMER$BASE$PORT+N*2;
456   2 =          OUTPUT(DATA$PORT)=LOW(VALUE);
457   2 =          CALL DELAY(2);
458   2 =          OUTPUT(DATA$PORT)=HIGH(VALUE);
459   2 =          RETURN;
460   2 =   TIMER$ERROR:    CALL SIO$NEWLINE;
461   2 =          CALL SIO$OUT$STRING(@('TIMER LOAD ERROR'),16);
462   2 =          END;
463   1 =   TIMER$READ:
        =          PROCEDURE(N) WORD PUBLIC;
464   2 =          DECLARE N WORD, DATA$PORT WORD, VALUE WORD, A BYTE;
465   2 =          IF N> 2 THEN GO TO TIME$ERROR;
467   2 =          DATA$PORT=TIMER$BASE$PORT+N*2;
468   2 =          A=INPUT(DATA$PORT);
469   2 =          VALUE=SHL(DOUBLE(INPUT(DATA$PORT)),8) + DOUBLE(A);
470   2 =          RETURN VALUE;
471   2 =   TIME$ERROR: CALL SIO$NEWLINE;
472   2 =          CALL SIO$OUT$STRING(@('TIMER READ ERROR'),16);
473   2 =          END;
        =
        =
            $INCLUDE (:F1:DIAGS.PLM)
```

```
          =   $SUBTITLE('DIAGNOSTIC MODULE')
          =       /* THIS MODULE CONTAINS TESTS WHICH ALLOW THE MANIPULATION
          =          OF VARIOUS SYSTEM COMPONENTS INCLUDING:
          =          1) MEMORY TEST (M)
          =          2) DAC RAMP TEST (R)
          =          3) DAC OUT/ ADC IN (L)
          =          4) DAC/ADC LOOP (H)
          =          5) CONSOLE LED/SWITCH (C)
          =          6) LSI CIRCUIT TEST (I)
          =          7) KEYBD/CONSOLE TEST (K)
          =          8) TIMER TEST(T)
          =          9) APU CONTROLLER TEST (B)
          =          10) APU TEST (A)
          =          11) DISCRETE I/O LOOP TEST (D)
          =          12) CONSOLE LED RAMP TEST (E)
          =          13) DISK IOPB/HARDWARE TEST (F)
          =       THE MODULE IS INVOKED FROM THE COMMAND MODE BY ENTERING T.
          =       A SECONDARY COMMAND IS REQUIRED TO ENTER A SPECIFIC TEST.
          =       CHARACTERS ARE INDICATED AT THE END OF THE LINES ABOVE.  */
   474  1 =       MEMORY$TEST:
          =           PROCEDURE /* THIS TEST DOES NOT GUARANTEE NO MEMORY FAULT
          =              IT IS MERELY MUCH LESS LIKELY */
          =           /* ROUTINE TESTS RAM MEMORY WITHIN A REQUESTED SEGMENT
          =           FOR FIRST FAILED LOCATION  */
   475  2 =           DECLARE TEST BYTE;
   476  2 =           CALL SIO$OUT$STRING(@('EMORY TEST'),10);
   477  2 =           CALL SIO$NEWLINE;
   478  2 =           CALL SIO$OUT$STRING(@('ENTER STARTING ADDRESS:'),23);
   479  2 =           CALL SIO$GET$CHAR;
   480  2 =           CALL SIO$GET$ADDR(0);
   481  2 =           TEST=TRUE;
   482  2 =           DO WHILE TEST=TRUE;
   483  3 =           MEMORY$ARG1=55H;/* WRITE ALTERNATING BITS (01010101) TO EACH LOCATION */
   484  3 =           CALL DELAY(1);
   485  3 =           IF MEMORY$ARG1 <> 55H THEN TEST=FALSE;/* VERIFY THE DATA WRITTEN */
   487  3 =           MEMORY$ARG1=0AAH;/* WRITE OPPOSITE PATTERN (10101010) */
   488  3 =           CALL DELAY(10);
   489  3 =           IF MEMORY$ARG1 <> 0AAH THEN TEST=FALSE;/* VERIFY DATA WRITTEN */
   491  3 =           ARG1.OFF = ARG1.OFF + 1;/* INCREMENT THE OFFSET */
   492  3 =           IF ARG1.OFF = 00000H THEN GO TO EXIT$DONE;/* VERIFY NOT AT THE MAX OFFSET */
   494  3 =           END;
   495  2 =           ARG1.OFF=ARG1.OFF-1;/* IF ERROR DECREMENT OFFSET AND PRINT WARNING */
   496  2 =           CALL SIO$NEWLINE;
   497  2 =           CALL SIO$OUT$STRING(@('MEMORY FAILED AT '),17);
   498  2 =           CALL SIO$OUT$WORD(ARG1.SEG);
   499  2 =           CALL SIO$OUT$CHAR(':');
   500  2 =           CALL SIO$OUT$WORD(ARG1.OFF);
   501  2 =           CALL SIO$OUT$STRING(@(' WITH THE VALUE '),16);
   502  2 =           CALL SIO$OUT$BYTE(MEMORY$ARG1);
   503  2 =           CALL SIO$NEWLINE;
   504  2 =       EXIT$DONE:  GO TO NEXT$COMMAND;
   505  2 =           END;
          =       /*                                                    */
   506  1 =       DAC$RAMP$TEST:
          =           PROCEDURE;/* STARTING FROM 0 VOLTS AND GOING TO +100 AND
          =              THEN TO -100 BACK TO ZERO FOR GIVEN DAC.  A RISING
```

```
        =               TRIANGULAR WAVE WITH NO IMPERFECTIONS SHOULD BE
        =               OBSERVED BY MONITORING */
  507   2 =           DECLARE I WORD, A WORD, BASE WORD;
  508   2 =           CALL SIO$OUT$STRING(@('AMP DACS'),8);
  509   2 =           A=0;
  510   2 =           CALL SIO$NEW$LINE;
  511   2 =           CALL SIO$OUT$STRING (@('ENTER PORT NUMBER:'),18);
  512   2 =           CALL SIO$GET$CHAR;
  513   2 =           BASE=SIO$GET$WORD;
  514   2 =           DO WHILE TRUE;
  515   3 =           OUTWORD(BASE)=A;
  516   3 =           A=A+1;/* DACS USED ARE 12 BIT */
  517   3 =           CALL DELAY(2);
  518   3 =           END;
  519   2 =           END;
  520   1 =   DAC$ADC$LOOP:
        =           PROCEDURE; /* TRANSFERS CH X ADC TO CH Y DAC */
  521   2 =           DECLARE INPORT WORD, OUTPORT WORD;
  522   2 =           CALL SIO$OUT$STRING(@('YBRID LOOP'),10);
  523   2 =           CALL SIO$NEW$LINE;
  524   2 =           CALL SIO$OUT$STRING(@('ENTER DAC PORT:'),15);
  525   2 =           CALL SIO$GET$CHAR;
  526   2 =           OUTPORT=SIO$GET$WORD;
  527   2 =           CALL SIO$NEWLINE;
  528   2 =           CALL SIO$OUT$STRING(@('ENTER ADC PORT:'),15);
  529   2 =           CALL SIO$GET$CHAR;
  530   2 =           INPORT=SIO$GET$WORD;
  531   2 =           DO WHILE TRUE;
  532   3 =           OUTWORD(OUTPORT)=INWORD(INPORT);
  533   3 =           CALL DELAY(2);
  534   3 =           END;
  535   2 =           END
  536   1 =   CONSOLE$LED$SWITCH$TEST:
        =           PROCEDURE;
        =               /* CONSOLE SWITCHES ARE SENT TO THE CONSOLE LEDS */
  537   2 =           CALL SIO$OUT$STRING(@('ONSOLE LEDS/SWITCHES'),20);
  538   2 =           DO WHILE TRUE;
  539   3 =           OUTWORD(CONSOLE$LED$PORT)=INWORD(CONSOLE$SWITCH$PORT);
  540   3 =           END;
  541   2 =           END;
  542   1 =   CONSOLE$LED$RAMP:
        =           PROCEDURE;
        =           /* CAUSES THE CONSOLE LEDS TO COUNT UP CONTINUOUSLY */
  543   2 =           DECLARE A WORD;
  544   2 =           CALL SIO$OUT$CHAR(7FH);
  545   2 =           CALL SIO$OUT$STRING(@('CONSOLE LED RAMP'),16);
  546   2 =           A=0;
  547   2 =           DO WHILE TRUE;
  548   3 =           OUTWORD(CONSOLE$LED$PORT)=A;
  549   3 =           CALL DELAY$LONG;
  550   3 =           A=A+1;
  551   3 =           END;
  552   2 =           END;
  553   1 =   DAC$OUT$ADC$IN:
        =           PROCEDURE;
        =           /* CAUSES A GIVEN CHANNEL DAC TO BE SET AND READ BY THE
```

```
                =           SAME CHANNEL ADC TO CHECK FOR SYSTEM ACCURACY  */
     554    2   =           DECLARE W WORD,B WORD, C WORD, T WORD, A WORD;
     555    2   =           CALL SIO$OUT$STRING(@('OOP TEST'),8);
     556    2   =           T=0020H; /* TOLERANCE VALUE */
     557    2   =       LOOPD:  CALL SIO$NEWLINE;
     558    2   =               CALL SIO$OUT$STRING(@('ENTER DAC VALUE: '),17);/* NUMBER IS
                =               ENTERED IN HEX, TWO'S COMPLEMENT */
                =               /* SAMPLE VALUES---
                =                   0000H -- 0.0 VOLTS
                =                   7FFFH -- 100.0 VOLTS
                =                   FFFFH -- SLIGHTLY NEGATIVE (LSB)
                =                   8000H -- -100.0 VOLTS
                =                   4000H -- 50.0 VOLTS */
     559    2   =           CALL SIO$GET$CHAR;
     560    2   =           W=SIO$GET$WORD;
     561    2   =           CALL SIO$NEWLINE;
     562    2   =           CALL SIO$OUT$STRING(@('ENTER PORT NUMBER: '),19);
     563    2   =           CALL SIO$GET$CHAR;
     564    2   =           B=SIO$GET$WORD;
     565    2   =           OUTWORD(B)=W;
     566    2   =           CALL DELAY$LONG;
     567    2   =           C=INWORD(B);
     568    2   =           A=W-C;
     569    2   =           IF A>127 THEN A=256-A;/* 127 IS MAX POSITIVE NUMBER
                =               STATEMENT TAKES ABSOLUTE VALUE */
     571    2   =           IF A> T THEN GO TO ERROR$DETECTED;
                =           ELSE
     573    2   =           GO TO LOOPD;
     574    2   =   ERROR$DETECTED:
                =           CALL SIO$NEWLINE;
     575    2   =           CALL SIO$OUT$STRING(@('HYBRID LOOP ERROR WAS GREATER THAN TOLERANCE--'),46);
     576    2   =           CALL SIO$OUT$WORD(A);
     577    2   =           GO TO LOOPD;
     578    2   =           END;
     579    1   =   APU$CONTROL$TEST:
                =           PROCEDURE;/* OUTPUTS A NUMBER FROM ENTERED LOCATION TO APU
                =               AND THEN RETRIEVES IT TO ANOTHER MEMORY LOCATION
                =               FOR VISUAL COMPARISON */
     580    2   =           DECLARE (A,B) POINTER;
     581    2   =           CALL SIO$OUT$CHAR(7FH);
     582    2   =           CALL SIO$OUT$STRING(@('APU CONTROL TEST'),16);
     583    2   =           CALL SIO$NEWLINE;
     584    2   =           CALL SIO$OUT$STRING(@('ENTER ADDRESS OF FLOATING POINT NUMBER '),39);
     585    2   =           CALL SIO$GET$CHAR;
     586    2   =           CALL SIO$GET$ADDR(0);
     587    2   =           A=MEMORY$ARG1$PTR;
     588    2   =           CALL ALU$OUT$WORD(MEMORY$ARG1$PTR);
     589    2   =           CALL SIO$NEWLINE;
     590    2   =           CALL SIO$OUT$STRING(@('ENTER TRANSFER ADDRESS '),23);
     591    2   =           CALL SIO$GET$CHAR;
     592    2   =           CALL SIO$GET$ADDR(0);
     593    2   =           B=MEMORY$ARG1$PTR;
     594    2   =           CALL ALU$IN$WORD(MEMORY$ARG1$PTR);
     595    2   =           CALL SIO$NEWLINE;
     596    2   =           CALL DISPLAY$DATA$ARRAY(A,4);/* DISPLAY ORIGINAL VALUE AND
                =               THE RETRIEVED VALUE */
```

```
597   2  =          CALL DISPLAY$DATA$ARRAY(B,4);
598   2  =          GO TO NEXT$COMMAND;
599   2  =          END;
600   1  =    LSI$CIRCUIT$TEST
              =          PROCEDURE;
              =          /* ALLOWS THE READING AND WRITING OF REGISTERS IN LSI CIRCUITS USING THE CONSOLE SWITCHES AND THE
               -    KEYBOARD */
601   2  =          DECLARE A WORD, B BYTE;
602   2  =          CALL SIO$OUT$STRING(@('NTEGRATED CIRCUITRY'),19);
603   2  =       AGAIN:  CALL SIO$NEWLINE;
604   2  =          CALL SIO$OUT$STRING(@('ENTER PORT NUMBER '),18);
605   2  =          CALL SIO$GET$CHAR;
606   2  =          A=SIO$GET$WORD;
607   2  =       REDO:   CALL SIO$NEWLINE;/* SET CONSOLE SWITCHES PRIOR TO WRITE */
608   2  =          CALL SIO$OUT$STRING(@('TYPE R TO READ, W TO WRITE '),27);
609   2  =          CALL SIO$GET$CHAR;/* REJECT IF CHAR IS NOT R OR W */
610   2  =          IF (CHAR <> 'R') AND (CHAR <> 'W') THEN GO TO REDO;
612   2  =          IF CHAR  = 'R' THEN OUTPUT(CONSOLE$LED$PORT)=INPUT(A);
              =          ELSE
614   2  =          OUTPUT(A)=INPUT(CONSOLE$SWITCH$PORT);
615   2  =          GO TO AGAIN;
616   2  =          END;
617   1  =    KEYBD$CONSOLE$TEST:
              =          PROCEDURE;
618   2  =          DECLARE TEST BYTE, I BYTE;
619   2  =          DECLARE ALPHA(*) BYTE DATA('ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789');
620   2  =          CALL SIO$OUT$STRING(@('EYBOARD DISPLAY TEST'),20);
621   2  =          CALL SIO$NEWLINE;
622   2  =          CALL SIO$OUT$STRING(@('ENTER THE EXACT SAME DATA'),25);
623   2  =          CALL SIO$NEWLINE;
624   2  =          CALL SIO$OUT$STRING(@ALPHA(0),36);
625   2  =          CALL SIO$NEWLINE;
626   2  =          TEST=TRUE;
627   2  =          I=0;
628   2  =          DO WHILE TEST=TRUE;
629   3  =          CALL USER$IO$GET$CHAR;
630   3  =          IF(CHAR<>ALPHA(I)) THEN TEST=FALSE;
632   3  =          I=I+1;
633   3  =          IF I=36 THEN RETURN;
635   3  =          END;
636   2  =          CALL SIO$NEWLINE;
637   2  =          CALL SIO$OUT$STRING(@('KEYBD ENTRY ERROR'),17);
638   2  =          CALL SIO$NEWLINE;
639   2  =          GO TO NEXT$COMMAND;
640   2  =          END;
641   1  =    TIMER$TEST:
              =          PROCEDURE;
642   2  =          CALL SIO$OUT$STRING(@('IMER TEST'),9);
643   2  =          CALL SIO$NEWLINE;
644   2  =          CALL TIMER$LOAD(1000,0);
645   2  =          CALL TIMER$LOAD(1000,1);
646   2  =          CALL TIMER$LOAD(1000,2);
647   2  =          END;
648   1  =    APU$TEST:
              =          PROCEDURE;
              =             /* CALCULATES THE SUM OF 2.0 AND 3.0 AND DISPLAYS
```

```
        =           THE RESULT TO CONSOLE */
649   2 =         DECLARE AA(4) BYTE DATA(02H,80H,00H,00H);
650   2 =         DECLARE BB(4) BYTE DATA(02H,0C0H,00H,00H);
651   2 =         DECLARE A(4) BYTE AT (00120H);
652   2 =         DECLARE B(4) BYTE AT (00124H);
653   2 =         DECLARE C(4) BYTE AT (00128H);
654   2 =         DECLARE I BYTE;
655   2 =         CALL SIO$OUT$STRING(@('PU TEST'),7);
656   2 =         CALL SIO$NEWLINE;
657   2 =         DO I = 0 TO 3;
658   3 =         A(I)=AA(I);
659   3 =         B(I)=BB(I);
660   3 =         END;
661   2 =         CALL ALU$OUT$WORD(@A);
662   2 =         CALL ALU$OUT$WORD(@B);
663   2 =         CALL ALU$CMND$WORD(10H);
664   2 =         CALL ALU$IN$WORD(@C);
665   2 =         MEMORY$ARG1$PTR=@A;
666   2 =         CALL SIO$CRLF;
667   2 =         CALL CONVERT$TO$FLOAT(0);
668   2 =         CALL SIO$OUT$STRING(@(' + '),3);
669   2 =         MEMORY$ARG1$PTR=@B;
670   2 =         CALL CONVERT$TO$FLOAT(0);
671   2 =         CALL SIO$OUT$CHAR('=');
672   2 =         MEMORY$ARG1$PTR=@C;
673   2 =         CALL CONVERT$TO$FLOAT(0);
674   2 =         CALL SIO$NEWLINE;
675   2 =         END;
676   1 =   DISCRETE$IO$TEST:
        =         PROCEDURE;
        =            /* CONNECT THE DISCRETE OUTPUTS TO THE
        =         DISCRETE INPUTS TO RUN THIS TEST. */
677   2 =         DECLARE A WORD;
678   2 =         CALL SIO$OUT$STRING(@('ISCRETE I/O TEST'),16);
679   2 =         CALL SIO$NEWLINE;
680   2 =         A=0;
681   2 =         DO WHILE A<>0FFFFH;
682   3 =         OUTWORD(DISCRETE$IO$PORT1)=A;
683   3 =         OUTWORD(DISCRETE$IO$PORT2)=A;
684   3 =         IF INWORD(DISCRETE$IO$PORT1)<>A THEN GO TO DISCRETE$ERROR;
686   3 =         IF INWORD(DISCRETE$IO$PORT2)<>A THEN GO TO DISCRETE$ERROR2;
688   3 =         A=A+1;
689   3 =         END;
690   2 =   DISCRETE$ERROR:
        =         CALL SIO$NEWLINE;
691   2 =         CALL SIO$OUT$STRING(@('DISCRETE TEST 1 FAILED--'),24);
692   2 =         CALL SIO$OUT$WORD(A);
693   2 =         GO TO NEXT$COMMAND;
694   2 =   DISCRETE$ERROR2:
        =         CALL SIO$NEWLINE;
695   2 =         CALL SIO$OUT$STRING(@('DISCRETE TEST 2 FAILED--'),24);
696   2 =         CALL SIO$OUT$WORD(A);
697   2 =         GO TO NEXT$COMMAND;
698   2 =         END;
699   1 =   DISK$TEST:
        =         PROCEDURE;
```

```
        =               /* ALLOWS THE TESTING OF THE DISK SUBSYSTEM
        =             BY DIRECT CONTROL THROUGH THE DISK DRIVER PACKAGE */
700   2 =             CALL SIO$OUT$STRING(@('LOPPY DISK'),10);
701   2 =             CALL SIO$NEWLINE;
702   2 =             CALL SIO$OUT$STRING(@('ENTER IOPB  '),12);
703   2 =             CALL SIO$GET$CHAR;
704   2 =             IOPB=SIO$GET$WORD;
705   2 =             CALL DISK;
706   2 =             END;
707   1 =   DIAGNOSTICS:
        =             PROCEDURE;
708   2 =             DECLARE SEC$COMMAND(*) BYTE DATA('MRHLCIKTBRDEF');
709   2 =             DECLARE (I,J) BYTE;
710   2 =             CALL SIO$OUT$STRING(@('EST MODULE'),10);
711   2 =             CALL PROMPT;
712   2 =             CALL SUPPRESS$BLANKS;
713   2 =             J=-1;
714   2 =             DO I=0 TO 12;
715   3 =             IF CHAR = SEC$COMMAND(I) THEN J=I;
717   3 =             END;
718   2 =             IF J>127 THEN GO TO ERROR;
720   2 =             DO CASE J;
721   3 =             CALL MEMORY$TEST;
722   3 =             CALL DAC$RAMP$TEST;
723   3 =             CALL DAC$ADC$LOOP;
724   3 =             CALL DAC$OUT$ADC$IN;
725   3 =             CALL CONSOLE$LED$SWITCH$TEST;
726   3 =             CALL LSI$CIRCUIT$TEST;
727   3 =             CALL KEYBD$CONSOLE$TEST;
728   3 =             CALL TIMER$TEST;
729   3 =             CALL APU$CONTROL$TEST;
730   3 =             CALL APU$TEST;
731   3 =             CALL DISCRETE$IO$TEST;
732   3 =             CALL CONSOLE$LED$RAMP;
733   3 =             CALL DISK$TEST;
734   3 =             END;
735   2 =             END;
```

```
                $SUBTITLE('COMMAND ROUTINES')
736   1         SIO$GO:
                    PROCEDURE;
737   2           CALL SIO$OUT$STRING(@('0 (ADRS)'),8);
738   2           CALL PROMPT;
739   2           CALL SUPPRESS$BLANKS;/* EXECUTE A PROGRAM AT ENTERED ADDRESS*/
740   2           CALL SIO$GET$ADDR(0);
741   2           IF CHAR<>ASCR THEN GOTO ERROR;
743   2           CALL SIO$CRLF;
744   2           CALL MEMORY$ARG1$PTR;/*EXECUTE IS ACTUALLY AN INDIRECT CALL */
745   2           END;
746   1         SIO$EXAM$MEM:
                    PROCEDURE;/* EXAMINE MEMORY AND ALLOW ALTERATION */
747   2           DECLARE T BYTE;
748   2           CALL SIO$OUT$STRING(@('UBSTITUTE (ADRS)'),16);
749   2           CALL PROMPT;
750   2           CALL SUPPRESS$BLANKS;
751   2           CALL SIO$GET$ADDR(0);/* WHAT ADDRESS? */
752   2           IF CHAR<>ASCR THEN GOTO ERROR;
754   2           DO WHILE TRUE;
755   3           CALL SIO$NEWLINE;
756   3           CALL SIO$OUT$WORD(ARG1.OFF);/* OUTPUT THE CURRENT VALUE */
757   3           CALL SIO$OUT$CHAR('-');
758   3           CALL SIO$OUT$BYTE(MEMORY$ARG1);
759   3           CALL SIO$OUT$CHAR('-');
760   3           CALL SIO$GET$CHAR;/* ALTER IF NEXT CHARACTER IS NOT CR,COMMA OR BLANK*/
761   3           IF CHAR<>',' AND CHAR<>ASBL AND CHAR<>ASCR THEN
762   3           DO;
763   4           T = SIO$GET$BYTE;
764   4           MEMORY$ARG1 = T;/* WRITE NEW VALUE AND VERIFY IT */
765   4           CALL DELAY(1);
766   4           IF MEMORY$ARG1<>T THEN DO;
768   5           CALL SIO$BLANKS(2);
769   5           CALL SIO$OUT$STRING(@('SUBSTITUTE FAILED'),17);
770   5           GO TO ERROR;
771   5           END;
772   4           END;
773   3           ARG1.OFF = ARG1.OFF+1;/* GO TO NEXT SEQUENTIAL LOCATION */
774   3           END;
775   2           END;
776   1         SIO$DISPLAY:
                    PROCEDURE;/* DISPLAY N WORDS STARTING AT ENTERED ADDRESS */
777   2           DECLARE COUNT WORD;
778   2           CALL SIO$OUT$STRING(@('IST (ADRS,CNT)'),14);
779   2           CALL PROMPT;
780   2           CALL SUPPRESS$BLANKS;
781   2           CALL SIO$GET$ADDR(0);/* WHAT ADDRESS? */
782   2           IF CHAR=ASCR THEN
783   2           COUNT  = 1;
                    ELSE
784   2           DO;
785   3           IF CHAR=':' THEN GOTO ERROR;
787   3           CALL SIO$GET$CHAR;
788   3           COUNT = SIO$GET$WORD;/* HOW MANY? ONE BY DEFAULT. */
789   3           IF CHAR<>ASCR THEN GOTO ERROR;
791   3           END;
```

```
792  2          CALL SIO$NEWLINE;
793  2          CALL SIO$OUT$WORD(ARG1.OFF);/* OUTPUT OFFSET AND THREE BLANKS*/
794  2          CALL SIO$BLANKS(3);
795  2          IF COUNT > 1 THEN
796  2          CALL SIO$BLANKS(LOW(3*(ARG1.OFF AND 000FH)));/* CENTER DATA */
797  2          DO WHILE COUNT > 0;
798  3          CALL SIO$OUT$BYTE(MEMORY$ARG1);/* OUTPUT THE DATA */
799  3          CALL SIO$OUT$CHAR(' ');
800  3          ARG1.OFF = ARG1.OFF + 1;
801  3          COUNT = COUNT - 1;
802  3          IF COUNT > 0 AND (ARG1.OFF AND 000FH)=0 THEN
803  3          DO;
804  4          CALL SIO$NEWLINE;
805  4          CALL SIO$OUT$WORD(ARG1.OFF);
806  4          CALL SIO$BLANKS(3);
807  4          END;
808  3          END;
809  2          END;
810  1      MEMORY$FILL:
               PROCEDURE;/* FILL MEMORY BEGINNING AT ENTERED ADDRESS FOR N LOCATIONS */
811  2          DECLARE COUNT WORD;
812  2          DECLARE T BYTE;
813  2          CALL SIO$OUT$STRING(@('ILL (START,CNT,VAL)'),19);
814  2          CALL PROMPT;
815  2          CALL SUPPRESS$BLANKS;
816  2          CALL SIO$GET$ADDR(0);
817  2          IF CHAR<>',' THEN GO TO ERROR;
819  2          CALL SIO$GET$CHAR;
820  2          COUNT=SIO$GET$WORD;/* HOW MANY BYTES DISPLAYED? */
821  2          IF CHAR<>',' THEN GO TO ERROR;
823  2          CALL SIO$GET$CHAR;
824  2          T=SIO$GET$BYTE;/* WITH WHAT VALUE? */
825  2          IF CHAR<>ASCR THEN GO TO ERROR;
827  2          DO WHILE COUNT>0;
828  3          CALL MOVB(@T,MEMORY$ARG1$PTR,1);
829  3          COUNT=COUNT-1;
830  3          ARG1.OFF=ARG1.OFF+1;
831  3          CALL DELAY(1);
832  3          END;
833  2          END;
834  1      MEMORY$MOVE:
               PROCEDURE;/* MOVE MEMORY FROM ADDRESS1 TO ADDRESS2 FOR T BYTES*/
835  2          DECLARE T WORD;
836  2          CALL SIO$OUT$STRING(@('OVE (SOURCE,DEST,CNT)'),21);
837  2          CALL PROMPT;
838  2          CALL SUPPRESS$BLANKS;
839  2          CALL SIO$GET$ADDR(0);
840  2          IF CHAR<>',' THEN GO TO ERROR;
842  2          CALL MOVB(@MEMORY$ARG1$PTR,@MEMORY$ARG2$PTR,4);/* USES SYSTEM MOVE ROUTINES*/
843  2          CALL SIO$GET$CHAR;
844  2          CALL SIO$GET$ADDR(0);
845  2          IF CHAR<>',' THEN GO TO ERROR;
847  2          CALL SIO$GET$CHAR;
848  2          T=SIO$GET$WORD;
849  2          IF CHAR<>ASCR THEN GO TO ERROR;
851  2          CALL MOVB(MEMORY$ARG2$PTR,MEMORY$ARG1$PTR,T);
```

```
852  2          END;
853  1      LIST$MEMORY
                PROCEDURE;/* DUMPS MEMORY FROM ADDRESS ENTERED FOR COUNT BYTES TO MICROPRINTER */
854  2          DECLARE COUNT WORD;
855  2          CALL SIO$OUT$STRING(@('RINT (ADRS,CNT)'),15);
856  2          CALL PROMPT;
857  2          CALL SUPPRESS$BLANKS;
858  2          CALL SIO$GET$ADDR(0);
859  2          IF CHAR=ASCR THEN COUNT=1;
                ELSE
861  2          DO;
862  3          IF CHAR =':' THEN GO TO ERROR;
864  3          CALL SIO$GET$CHAR;
865  3          COUNT=SIO$GET$WORD;
866  3          IF CHAR<>ASCR THEN GO TO ERROR;
868  3          END;
869  2          CALL LP$NEWLINE;
870  2          CALL LP$OUT$STRING(@('8086 MEMORY DUMP'),16);
871  2          CALL LP$NEWLINE;
872  2          CALL LP$NEWLINE;
873  2          CALL LP$OUT$WORD(ARG1.OFF);
874  2          CALL LP$BLANKS(3);
875  2          IF COUNT >1 THEN CALL LP$BLANKS(LOW(3*(ARG1.OFF AND 000FH)));
877  2          DO WHILE COUNT >0;
878  3          CALL LP$OUT$BYTE(MEMORY$ARG1);
879  3          CALL LP$OUT$CHAR(' ');
880  3          ARG1.OFF=ARG1.OFF+1;
881  3          COUNT=COUNT-1;
882  3          IF COUNT >0 AND (ARG1.OFF AND 000FH) = 0 THEN
883  3          DO;
884  4          CALL LP$NEWLINE;
885  4          CALL LP$OUT$WORD(ARG1.OFF);
886  4          CALL LP$BLANKS(3);
887  4          END;
888  3          END;
889  2          CALL LP$NEWLINE;
890  2          END;
891  1      BREAK$POINT$SET:
                PROCEDURE;/* ALLOWS A SINGLE BREAKPOINT TO BE SET ANYWHERE IN RAM. USES
            THE INTERRUPT 3 CAPABILITY OF THE SYSTEM */
892  2          CALL SIO$OUT$STRING(@('REAKPOINT SET (ADRS)'),20);
893  2          CALL PROMPT;
894  2          IF BREAK$POINT$FLAG <>0 THEN GO TO BREAK$POINT$ABORT;/* ONE ONLY */
896  2          CALL SIO$GET$CHAR;
897  2          DO WHILE CHAR=ASBL;
898  3          CALL SIO$GET$CHAR;
899  3          END;
900  2          CALL SIO$GET$ADDR(0);
901  2          IF CHAR<>ASCR THEN GO TO ERROR;
903  2          CALL MOVB(@MEMORY$ARG1$PTR,@BREAK$POINT$PTR,4);/* SAVE ADDRESS */
904  2          CALL MOVB(BREAK$POINT$PTR,@BREAK$POINT$DATA,2);/* SAVE ORIGINAL DATA */
905  2          CALL MOVB(@(0CDH,03H),BREAK$POINT$PTR,2);/* INSERT INTERRUPT CODE*/
906  2          BREAK$POINT$FLAG=1;/* SET FLAG INDICATING BREAKPOINT ACTIVE */
907  2          RETURN;
908  2      BREAK$POINT$ABORT:
                    /* THIS ROUTINE WARNS IF A BREAKPOINT IS ALREADY SET AND
```

```
                THE USER ATTEMPTS TO SET AN ADDITIONAL BREAKPOINT */
                    CALL SIO$NEWLINE.
 909    2           CALL SIO$OUT$STRING(@('BREAK POINT ALREADY SET AT '),27);
 910    2           CALL SIO$OUT$WORD(BKPTR.SEG);
 911    2           CALL SIO$OUT$CHAR(':');
 912    2           CALL SIO$OUT$WORD(BKPTR.OFF);
 913    2           CALL SIO$NEWLINE;
 914    2           END;
 915    1       BREAK$POINT$CLEAR.
                        /* THIS ROUTINE ALLOWS THE REMOVAL OF A BREAKPOINT WHICH
                HAS NOT BEEN ENCOUNTERED */
                    PROCEDURE;
 916    2           CALL SIO$OUT$STRING(@('LEAR BREAKPOINT'),15);
 917    2           CALL SIO$NEWLINE;
 918    2           CALL MOVB(@BREAK$POINT$DATA,BREAK$POINT$PTR,2);/* RESTORE DATA*/
 919    2           BREAK$POINT$FLAG=0;/* RESET BREAKPOINT FLAG */
 920    2           END;
```

```
                $SUBTITLE('INTERRUPT SERVICE ROUTINES')
 921   1        BREAK$POINT$SERVICE:
                PROCEDURE INTERRUPT 3;
 922   2        CALL BREAK$POINT$CLEAR;/* REMOVE THE BREAKPOINT*/
 923   2        CALL SIO$NEWLINE;
                   /* GIVE USER AN INDICATION BREAKPOINT WAS REACHED */
 924   2        CALL SIO$OUT$STRING(@('BREAK POINT ENCOUNTERED AT '),27);
 925   2        CALL SIO$OUT$WORD(BKPTR.SEG);
 926   2        CALL SIO$OUT$CHAR(':');
 927   2        CALL SIO$OUT$WORD(BKPTR.OFF);
 928   2        CALL SIO$NEWLINE;
 929   2        GO TO NEXT$COMMAND;
 930   2        END;
 931   1        DIVIDE$ERROR:
                 PROCEDURE INTERRUPT 0;/* SYSTEM ARITHMETIC INTERRUPT */
 932   2        CALL SIO$OUT$STRING(@('DIVIDE ERROR'), 12);
 933   2        GO TO NEXT$COMMAND;
 934   2        END;
 935   1      NMI:
                 PROCEDURE INTERRUPT 2;/* NON-MASKABLE INTERRUPT SERVICE */
 936   2        GO TO NEXT$COMMAND;
 937   2        RETURN;
 938   2        END;
 939   1      OVERFLOW$ERROR:
                 PROCEDURE INTERRUPT 4;/* SYSTEM ARITHMETIC INTERRUPT */
 940   2        CALL SIO$OUT$STRING(@('OVERFLOW'), 8);
 941   2        GOTO NEXT$COMMAND;
 942   2        END;
 943   1        INT20: PROCEDURE INTERRUPT 20;/* SERVICE FOR 8259A SYSTEM INTERRUPTS */
 944   2        CALL SIO$OUT$STRING(@('INTERRUPT 20'), 12);
 945   2        CALL SIO$NEWLINE;
 946   2        RETURN;
 947   2        END;
 948   1        INT21: PROCEDURE INTERRUPT 21;
 949   2        CALL SIO$OUT$STRING(@('INTERRUPT 21'), 12);
 950   2        CALL SIO$NEWLINE;
 951   2        RETURN;
 952   2      ;
 953   2        END;
 954   1        INT22: PROCEDURE INTERRUPT 22;
 955   2        CALL SIO$OUT$STRING(@('INTERRUPT 22'), 12);
 956   2        CALL SIO$NEWLINE;
 957   2        RETURN;
 958   2        END;
 959   1        INT23: PROCEDURE INTERRUPT 23;
 960   2        CALL SIO$OUT$STRING(@('INTERRUPT 23'), 12);
 961   2        CALL SIO$NEWLINE;
 962   2        RETURN;
 963   2        END;
 964   1        INT24: PROCEDURE INTERRUPT 24;
 965   2        CALL SIO$OUT$STRING(@('INTERRUPT 24'), 12);
 966   2        CALL SIO$NEWLINE;
 967   2        RETURN;
 968   2
 969   2        END;
 970   1        INT25: PROCEDURE INTERRUPT 25;
```

```
971   2         CALL SIO$OUT$STRING(@('INTERRUPT 25'), 12);
972   2         CALL SIO$NEWLINE;
973   2         RETURN;
974   2         END;
975   1         INT26: PROCEDURE INTERRUPT 26;
976   2         CALL SIO$OUT$STRING(@('INTERRUPT 26'), 12);
977   2         CALL SIO$NEWLINE;
978   2         RETURN;
979   2         END;
980   1         INT27: PROCEDURE INTERRUPT 27;
981   2         CALL SIO$OUT$STRING(@('INTERRUPT 27'), 12);
982   2         CALL SIO$NEWLINE;
983   2         RETURN;
984   2         END;
985   1         INIT$INT$VECTOR:
                  PROCEDURE(INT$VECT$PTR, INT$OFFSET);
986   2           DECLARE INT$VECT$PTR POINTER,
                      INT$OFFSET WORD,
                      VECTOR BASED INT$VECT$PTR STRUCTURE
                      (OFF WORD, SEG WORD),
                      CORRECTION LITERALLY '19H',
                    INT$INIT$VECT$CODE(*) BYTE DATA
                    (55H, 8BH, 0ECH, 8CH, 0C8H, 0C4H, 5EH, 04H, 26H, 89H, 07H,
                     5DH, 0C2H, 04H, 00H),
                    INIT$INT$VECT$CODE$PTR WORD
                    DATA ( INT$INIT$VECT$CODE);
987   2           CALL INIT$INT$VECT$CODE$PTR(@VECTOR.SEG);
988   2           VECTOR.OFF=INT$OFFSET-CORRECTION;
989   2         END;
```

```
                $SUBTITLE( COMMAND DISPATCH MODULE )
 990   1          GET$CMND:
                   PROCEDURE BYTE;
 991   2          DECLARE I BYTE;
 992   2          CALL SIO$NEWLINE;/* GIVE USER  A PROMPT */
 993   2          CALL SIO$OUT$STRING(@('ENTER COMMAND PLEASE:'),21);
 994   2          CALL SIO$GET$CHAR;
 995   2          DO I=0 TO 17;/* DECODE A COMMAND ENTRY */
 996   3          IF CHAR=SIO$CMND(I) THEN RETURN I;
 998   3          END;
 999   2          GOTO ERROR;/* BOMB IF NOT IN COMMAND TABLE */
1000   2          END;
                      /* BOOTSTRAP ENTRY */
1001   1          MAIN$PROGRAM:
                      /* SET STACK POINTER */
                   STACKPTR=0200H;
                      /* DELAY TO ALLOW VIDEO UNIT TO SELF PROGRAM */
1002   1          CALL INIT$INT$VECTOR(@INT$VECTOR(0),.DIVIDE$ERROR);
1003   1          CALL INIT$INT$VECTOR(@INT$VECTOR(2),.NMI);
1004   1          CALL INIT$INT$VECTOR(@INT$VECTOR(3),.BREAK$POINT$SERVICE);
1005   1          CALL INIT$INT$VECTOR(@INT$VECTOR(4),.OVERFLOW$ERROR);
1006   1          CALL INIT$INT$VECTOR(@INT$VECTOR(20),.INT20);
1007   1          CALL INIT$INT$VECTOR(@INT$VECTOR(21),.INT21);
1008   1          CALL INIT$INT$VECTOR(@INT$VECTOR(22),.INT22);
1009   1          CALL INIT$INT$VECTOR(@INT$VECTOR(23),.INT23);
1010   1          CALL INIT$INT$VECTOR(@INT$VECTOR(24),.INT24);
1011   1          CALL INIT$INT$VECTOR(@INT$VECTOR(25),.INT25);
1012   1          CALL INIT$INT$VECTOR(@INT$VECTOR(26),.INT26);
1013   1          CALL INIT$INT$VECTOR(@INT$VECTOR(27),.INT27);
1014   1          CALL DELAY$LONG;
1015   1          CALL DELAY$LONG;
1016   1          CALL DELAY$LONG;
1017   1          CALL DELAY$LONG;
                      /* INITIALIZE THE INTERRUPT CONTROLLER */
1018   1          CALL INT$INIT;
                     /* INITIALIZE THE USART */
1019   1          CALL SIO$INIT;
                      /* INITIALIZE THE MICROPRINTER */
1020   1          CALL LP$INIT;
                      /*INITIALIZE THE TIMER */
1021   1          CALL TIMER$INIT;
1022   1          CALL CRT$OUT(0CH);
1023   1          CALL CRT$OUT(0CH); /* CLEAR CRT CONSOLE */
1024   1          BREAK$POINT$FLAG=0;/* CLEAR THE BREAKPOINT FLAG */
1025   1          CALL CONSOLE$IDENT;
1026   1          CALL SIO$NEWLINE;/* OUTPUT THE SYSTEM LOGO */
1027   1          CALL SIO$OUT$STRING(@('USAFA 8086 VER 3.20'),19);
1028   1          CALL SIO$OUT$STRING(@(0DH,0AH,'DISK 0 '),9);
1029   1          OUTPUT(DISK$DRIVE$PORT)=01H; /* SELECT DRIVE 0 */
1030   1          OUTPUT(DISK$RESET$PORT)=00H; /* RESET DISK CONTROLLER */
1031   1          CALL DISK$BUSY;
1032   1          OUTPUT(DISK$COMMAND$PORT)=0F0H;/*RESTORE DRIVE 0 */
1033   1          CALL DISK$BUSY;
1034   1          IF((NOT(INPUT(DISK$STATUS$PORT))) AND DISK$READY)<>00H)
                      THEN CALL SIO$OUT$STRING(@('NOT '),4);
1036   1          CALL SIO$OUT$STRING(@('READY'),5);
```

```
1037  1          CALL SIO$OUT$STRING(@(0DH,0AH,'DISK 1 '),9);
1038  1          OUTPUT(DISK$DRIVE$PORT)=02H; /* SELECT DRIVE 1 */
1039  1          OUTPUT(DISK$RESET$PORT)=00H; /* RESET CONTROLLER */
1040  1          CALL DISK$BUSY;
1041  1          OUTPUT(DISK$COMMAND$PORT)=0F0H; /*RESTORE DRIVE 1*/
1042  1          CALL DISK$BUSY;
1043  1          IF(((NOT(INPUT(DISK$STATUS$PORT))) AND DISK$READY)<>00H)
                      THEN CALL SIO$OUT$STRING(@('NOT '),4);
1045  1          CALL SIO$OUT$STRING(@('READY'),5);
1046  1          DISK$TRACK(0)=00H;
1047  1          DISK$TRACK(1)=00H;
1048  1      NEXT$COMMAND:
                  DO WHILE TRUE;
1049  2          STACKPTR = 0200H;/* REINITIALIZE THE STACK */
1050  2          DO CASE GET$CMND;/* GET COMMAND FROM THE CONSOLE AND EXECUTE */
1051  3          CALL SIO$GO;
1052  3          CALL SIO$EXAM$MEM;
1053  3          CALL SIO$DISPLAY;
1054  3          CALL MEMORY$MOVE;
1055  3          CALL MEMORY$FILL;
1056  3          CALL LIST$MEMORY;
1057  3          CALL BREAK$POINT$SET;
1058  3          CALL BREAK$POINT$CLEAR;
1059  3          CALL CONVERT$TO$FLOAT(1);
1060  3          CALL DIAGNOSTICS;
1061  3          CALL READ$DRIVER;
1062  3          CALL WRITE$DRIVER;
1063  3          CALL KOPY$DRIVER;
1064  3          CALL RENAME$DRIVER;
1065  3          CALL ERASE$DRIVER;
1066  3          CALL ATTRIBUTE$CHANGE$DRIVER;
1067  3          CALL INITIALIZE;
1068  3          CALL DIRECTORY$LIST;
1069  3          END;
1070  2          END;
1071  1      ERROR:
                  STACKPTR=0200H;/* CLEAR THE STACK */
1072  1          CALL SIO$NEWLINE;/* ISSUE CONSOLE WARNING  */
1073  1          CALL SIO$OUT$STRING(@('COMMAND ENTRY ERROR, TRY AGAIN'),30);
1074  1          CALL SIO$NEWLINE;
1075  1          GOTO NEXT$COMMAND;/* GET A NEW COMMAND */
1076  1          END USAF86;
```

CROSS-REFERENCE LISTING

DEFN	ADDR	SIZE	NAME, ATTRIBUTES, AND REFERENCES	
464	0069H	1	A	BYTE 468 469
651	0120H	4	A	BYTE ARRAY(4) AT ABSOLUTE 658 661 665
272	000AH	4	A	POINTER PARAMETER AUTOMATIC 273 274 282
507	0024H	2	A	WORD 509 515 516
601	0040H	2	A	WORD 606 613 614
677	0042H	2	A	WORD 680 681 682 683 684 686 688 692 696
580	0038H	4	A	POINTER 587 596
543	002CH	2	A	WORD 546 548 550
554	0036H	2	A	WORD 568 569 570 571 576
649	0054H	4	AA	BYTE ARRAY(4) DATA 658
603	1700H		AGAIN	LABEL 615
346	0114H	4	ALMOSTTEN	BYTE ARRAY(4) AT ABSOLUTE 367 372
347	0028H	4	ALMOSTTENS	BYTE ARRAY(4) DATA 367
619	0030H	36	ALPHA	BYTE ARRAY(36) DATA 624 630
6			ALUAUTOPORT	LITERALLY 323 330
332	0F3EH	19	ALUCMNDWORD	PROCEDURE PUBLIC STACK=0004H 374 380 389 396 404 405 406 407 410 411 663

```
     6                  ALUCONTROLPORT          LITERALLY
                                                 334  337

     6                  ALUDATAPORT             LITERALLY

   325  0F10H    46     ALUINWORD               PROCEDURE PUBLIC STACK=0006H
                                                 375  381  390  397  408  594  664

   318  0EE5H    43     ALUOUTWORD              PROCEDURE PUBLIC STACK=0006H
                                                 372  373  378  379  387  388  394  395  399  403  588  661  662

   336  0F51H    14     ALUSTATUSWORD           PROCEDURE BYTE PUBLIC STACK=0002H

   579  1648H   166     APUCONTROLTEST          PROCEDURE STACK=003EH
                                                 729

   648  1842H   239     APUTEST                 PROCEDURE STACK=0044H
                                                 730

    15  0000H     4     ARG1                    STRUCTURE AT
                                                 242  243  246  247  360  361  491  492  495  498  500  756  773  793
                                                 796  800  802  805  830  873  876  880  882  885

    16  0004H     4     ARG2                    STRUCTURE AT

    14                   ASBL                  LITERALLY
                                                  78  212  229  253  306  761  897

    13  0014H    16     ASCII                   BYTE ARRAY(16) DATA
                                                  61   62  166  167  417  420  430  431

    14                   ASCR                  LITERALLY
                                                  82  212  229  257  741  752  761  782  789  825  849  859  866  901

    14                   ASLF                  LITERALLY
                                                 259

    14                   ASTERISK              LITERALLY
                                                 270

    40  0000H           ATTRIBUTECHANGEDRIVER   PROCEDURE EXTERNAL(7) STACK=0000H
                                                1066

   332  0000H     1  B    B                     BYTE PARAMETER AUTOMATIC
                                                 333  334

   500  003CH     4  B                          POINTER
                                                 593  597

   601  0068H     1  B                          BYTE

   554  0030H     2  B                          WORD
                                                 564  565  567

   652  0124H     4  B                          BYTE ARRAY(4) AT ABSOLUTE
                                                 659  662  669
```

```
274  0000H    1  B                    BYTE BASED(A) ARRAY(1)
                                       282

204  0056H    1  B                    BYTE
                                       207  209  213

 59  0000H    1  B                    BYTE PARAMETER AUTOMATIC
                                        60   61   62

164  0000H    1  B                    BYTE PARAMETER AUTOMATIC
                                       165  166  167

507  0026H    2  BASE                 WORD
                                       513  515

650  0058H    4  BB                   BYTE ARRAY(4) DATA
                                       659

 17  0000H    4  BKPTR                STRUCTURE AT
                                       910  912  925  927

 17  0000H    1  BREAKPOINT           BYTE BASED(BREAKPOINTPTR)

908  1F0BH       BREAKPOINTABORT      LABEL
                                       895

915  1F40H   44  BREAKPOINTCLEAR      PROCEDURE STACK=0030H
                                       922 1058

 10  004DH    2  BREAKPOINTDATA       BYTE ARRAY(2)
                                       904  918

 11  004FH    1  BREAKPOINTFLAG       BYTE
                                       894  906  919 1024

 17  0000H    4  BREAKPOINTPTR        POINTER
                                        17  903  904  905  918

921  1F85H   85  BREAKPOINTSERVICE    PROCEDURE INTERRUPT(3) STACK=0034H
                                      1004

891  1E92H  174  BREAKPOINTSET        PROCEDURE STACK=003AH
                                      1057

405  1192H       BYPASSMULTIPLY       LABEL
                                       402

653  0128H    4  C                    BYTE ARRAY(4) AT ABSOLUTE
                                       664  672

 51  0000H    1  C                    BYTE PARAMETER AUTOMATIC
                                        52   55   56   57

196  0000H    1  C                    BYTE PARAMETER AUTOMATIC
                                       197  198  199  200  201
```

```
125  0008H    1  C                    BYTE PARAMETER AUTOMATIC
                                       126  129

157  0008H    1  C                    BYTE PARAMETER AUTOMATIC
                                       158  160  162

131  0008H    1  C                    BYTE PARAMETER AUTOMATIC
                                       132  133  134  136  137  139

 88  0008H    1  C                    BYTE PARAMETER AUTOMATIC
                                        89   90   91

554  0032H    2  C                    WORD
                                       567  568

  3  004CH    1  CHAR                 BYTE PUBLIC
                                       146  151  154  155  186  187  205  208  209  212  222  223  229  231
                                       233  244  299  301  302  306  316  610  612  630  715  741  752  761
                                       782  785  789  817  821  825  840  845  849  859  862  866  897  901
                                       996

 12  0050H    1  CONSOLE              BYTE
                                       159  161  181  183  258  260  262  291  292  294  296  298  308  311
                                       313

290  0E4DH   78  CONSOLEIDENT         PROCEDURE STACK=0020H
                                      1025

 19              CONSOLELEDPORT       LITERALLY
                                       441  539  548  613

542  1564H   51  CONSOLELEDRAMP       PROCEDURE STACK=002EH
                                       732

536  154AH   26  CONSOLELEDSWITCHTEST  PROCEDURE STACK=002EH
                                       725

 19              CONSOLESWITCHPORT    LITERALLY
                                       437  539  614

339  0F5FH  868  CONVERTTOFLOAT        PROCEDURE PUBLIC STACK=003CH
                                       667  670  673 1059

986              CORRECTION           LITERALLY
                                       988

854  004AH    2  COUNT                WORD
                                       860  865  875  877  881  882

777  0044H    2  COUNT                WORD
                                       783  788  795  797  801  802

811  0046H    2  COUNT                WORD
                                       820  827  829

350  005BH    1  COUNT                BYTE
```

```
                                        359  386  393  422  423  424

  131  097DH  155  CRTOUT               PROCEDURE PUBLIC STACK=000EH
                                        162  261  263 1022 1023

   18              CRTPORT             LITERALLY
                                        134  136

  520  14E0H  106  DACADCLOOP           PROCEDURE STACK=0030H
                                        723

  553  1597H  177  DACOUTADCIN          PROCEDURE STACK=0034H
                                        724

  506  148EH   82  DACRAMPTEST          PROCEDURE STACK=0030H
                                        722

  452  001CH    2  DATAPORT             WORD
                                        455  456  458

  464  001EH    2  DATAPORT             WORD
                                        467  468  469

  240  0000H    2  DEFAULTBASE          WORD PARAMETER AUTOMATIC
                                        241  242

   88  0058H   26  DELAY                PROCEDURE PUBLIC STACK=0004H
                                        101  109  111  116  118  120  122  135  141  445  447  449  457  484
                                        488  517  533  765  831

   94  00C2H   41  DELAYLONG             PROCEDURE PUBLIC STACK=0002H
                                        140  549  566 1014 1015 1016 1017

   23  0051H    1  DERR                 BYTE

  707  19F2H  185  DIAGNOSTICS           PROCEDURE STACK=0048H
                                        1060

   44  0000H       DIRECTORYLIST        PROCEDURE EXTERNAL(9) STACK=0000H
                                        1068

  690  1973H       DISCRETEERROR        LABEL
                                        685

  694  1993H       DISCRETEERROR2       LABEL
                                        687

   21              DISCRETEIOPORT1      LITERALLY
                                        682  684

   21              DISCRETEIOPORT2      LITERALLY
                                        683  686

  676  1931H  130  DISCRETEIOTEST       PROCEDURE STACK=0034H
                                        731
```

```
  26  0000H        DISK                PROCEDURE EXTERNAL(1) STACK=0000H
                                        705

 100  08EBH   31   DISKBUSY            PROCEDURE STACK=000CH
                                        1031 1033 1040 1042

  29                DISKCOMMANDPORT     LITERALLY
                                        1032 1041

  29                DISKDRIVEPORT       LITERALLY
                                        1029 1038

  29                DISKREADY           LITERALLY
                                        102 1034 1043

  29                DISKRESETPORT       LITERALLY
                                        1030 1039

  29                DISKSTATUSPORT      LITERALLY
                                        102  104 1034 1043

 699  1983H   63   DISKTEST            PROCEDURE STACK=0030H
                                        733

  28  0031H    4   DISKTRACK           BYTE ARRAY(4) PUBLIC AT ABSOLUTE
                                        1046 1047

 351  0110H    4   DISPLAYCHAR         BYTE ARRAY(4) AT ABSOLUTE
                                        408  409

 272  0DDFH  110   DISPLAYDATAARRAY     PROCEDURE PUBLIC STACK=0036H
                                        596  597

 931  1FDAH   46   DIVIDEERROR         PROCEDURE INTERRUPT(0) STACK=002EH
                                        1002

 392  1117H        DIVIDELOOP          LABEL
                                        377

                   DOUBLE              BUILTIN
                                        223  469

  38  0000H        ERASEDRIVER         PROCEDURE EXTERNAL(6) STACK=0000H
                                        1065

  25  0764H        ERROR               LABEL PUBLIC
                                        206  214  236  719  742  753  770  786  790  818  822  826  841  846
                                        850  863  867  902  999 1071

 574  1628H        ERRORDETECTED       LABEL
                                        572

 504  148BH        EXITDONE            LABEL
                                        493

 353  0064H    1   EXP1                BYTE
```

```
                                        422  431

353  0065H    1  EXP2                   BYTE
                                         424  430

353  0066H    1  EXPSIGN                BYTE
                                         385  392  427

  4                FALSE                LITERALLY
                                         194  486  490  631

359  0F82H        FLOATINGDISPLAY        LABEL
                                         356

341  0018H    4  FLTADR                 STRUCTURE AT
                                         360  361

325  0000H    4  FLTADR                 POINTER PARAMETER AUTOMATIC
                                         326  328

318  0000H    4  FLTADR                 POINTER PARAMETER AUTOMATIC
                                         319  321

341  0018H    4  FLTADRS                POINTER
                                         341  362  365

327  0014H    4  FLTADRS                STRUCTURE AT
                                         329  330

320  0010H    4  FLTADRS                STRUCTURE AT
                                         322  323

341  0000H    4  FLTDATA                BYTE BASED(FLTADRS) ARRAY(4)
                                         362  365

327  0014H    4  FLTPTR                 POINTER
                                         327  328

320  0010H    4  FLTPTR                 POINTER
                                         320  321

153  0A4AH   45  FORCEUPPERCASE         PROCEDURE PUBLIC STACK=0004H
                                         185  315

 23  0000H    1  FTERR                  BYTE EXTERNAL(0)

990  21EDH   73  GETCMND                PROCEDURE BYTE STACK=0030H
                                        1050

190  0008H    1  H                      BYTE PARAMETER AUTOMATIC
                                         191  192

                 HIGH                   BUILTIN
                                          66  171  458

353  0068H    1  I                      BYTE
```

```
                                        364  365  366  367  368  400  401  409  419  420

      991  0073H     1  I                BYTE
                                         995  996  997

      273  0059H     1  I                BYTE
                                         280  282

      251  0058H     1  I                BYTE
                                         252

      654  006EH     1  I                BYTE
                                         657  658  659

      709  006FH     1  I                BYTE
                                         714  715  716

       95  0054H     1  I                BYTE
                                          96

       70  0052H     1  I                BYTE
                                          71   72

      175  0055H     1  I                BYTE
                                         176  177

      618  006DH     1  I                BYTE
                                         627  630  632  633

       76  0053H     1  I                BYTE
                                          77

      507  0022H     2  I                WORD

       42  0000H        INITIALIZE        PROCEDURE EXTERNAL(8) STACK=0000H
                                         1067

      986  0000H     2  INITINTVECTCODEPTR  WORD DATA
                                         987

      985  21CCH    33  INITINTVECTOR     PROCEDURE STACK=000EH
                                         1002 1003 1004 1005 1006 1007 1008 1009 1010 1011 1012 1013

      521  0020H     2  INPORT            WORD
                                         530  532

                        INPUT             BUILTIN
                                          53  102  104  127  144  146  149  151  293  295  299  301  337  468
                                         469  613  614 1034 1043

      943  2055H    50  INT20             PROCEDURE INTERRUPT(20) STACK=0030H
                                         1006

      948  2087H    50  INT21             PROCEDURE INTERRUPT(21) STACK=0030H
                                         1007
```

```
954  20B9H   50  INT22                PROCEDURE INTERRUPT(22) STACK=0030H
                                       1008

959  20EBH   50  INT23                PROCEDURE INTERRUPT(23) STACK=0030H
                                       1009

964  211DH   50  INT24                PROCEDURE INTERRUPT(24) STACK=0030H
                                       1010

970  214FH   50  INT25                PROCEDURE INTERRUPT(25) STACK=0030H
                                       1011

975  2181H   50  INT26                PROCEDURE INTERRUPT(26) STACK=0030H
                                       1012

980  21B3H   50  INT27                PROCEDURE INTERRUPT(27) STACK=0030H
                                       1013

  8              INTICW1              LITERALLY
                                       108

  8              INTICW2              LITERALLY
                                       110

  8              INTICW4              LITERALLY
                                       112

107  0900H   33  INTINIT              PROCEDURE STACK=000CH
                                       1018

986  0069H   15  INTINITVECTCODE      BYTE ARRAY(15) DATA
                                       986

  8              INTMASKPORT          LITERALLY
                                       110  112

985  0004H    2  INTOFFSET            WORD PARAMETER AUTOMATIC
                                       986  988

  8              INTSTATPORT          LITERALLY
                                       108

  2  0000H  400  INTVECTOR            POINTER ARRAY(100) AT ABSOLUTE
                                       1002 1003 1004 1005 1006 1007 1008 1009 1010 1011 1012 1013

985  0006H    4  INTVECTPTR           POINTER PARAMETER AUTOMATIC
                                       986  987  988

                 INWORD               BUILTIN
                                       437  532  539  567  684  686

 22  0036H    2  IOPB                 WORD PUBLIC AT ABSOLUTE
                                       704

709  0070H    1  J                    BYTE
                                       713  716  718  720
```

```
148  0A31H   25  KEYBDCHARIN          PROCEDURE PUBLIC STACK=0002H
                                       184  314

617  176FH  147  KEYBDCONSOLETEST     PROCEDURE STACK=0030H
                                       727

 18              KEYBDDATAPORT        LITERALLY
                                       151  301

 18              KEYBDRDY             LITERALLY
                                       149  295

 18              KEYBDSTATPORT        LITERALLY
                                       149  295

 34  0000H       KOPYDRIVER           PROCEDURE EXTERNAL(4) STACK=0000H
                                      1063

853  1D98H  250  LISTMEMORY           PROCEDURE STACK=003AH
                                      1056

557  15AFH       LOOPD                LABEL
                                       573  577

                 LOW                  BUILTIN
                                        67  172  456  796  876

 75  0851H   42  LPBLANKS             PROCEDURE PUBLIC STACK=0010H
                                        86  874  876  886

 81  087BH   20  LPCRLF               PROCEDURE PUBLIC STACK=000EH
                                        85

  9              LPDATAPORT           LITERALLY
                                        47   48   49   55   56   57

 46  078FH   15  LPINIT               PROCEDURE STACK=0002H
                                      1020

 84  088FH   25  LPNEWLINE            PROCEDURE PUBLIC STACK=0018H
                                       869  871  872  884  889

 59  07C4H   52  LPOUTBYTE            PROCEDURE PUBLIC STACK=0010H
                                        66   67  878

 51  079EH   38  LPOUTCHAR            PROCEDURE PUBLIC STACK=0006H
                                        61   62   72   78   82  879

 69  081AH   55  LPOUTSTRING          PROCEDURE PUBLIC STACK=0014H
                                       870

 64  07F8H   34  LPOUTWORD            PROCEDURE PUBLIC STACK=001AH
                                       873  885

  9              LPSTATUSPORT         LITERALLY
                                        53
```

```
600  16EEH  129  LSICIRCUITTEST        PROCEDURE STACK=0030H
                                       726

273  005AH    1  M                    BYTE
                                       277  278  279  280  286

 24  04E1H       MAINPROGRAM          LABEL PUBLIC
                                       1001

 15  0000H    1  MEMORYARG1           BYTE BASED(MEMORYARG1PTR)
                                       483  485  487  489  502  758  764  766  798  878

 15  0000H    4  MEMORYARG1PTR        POINTER PUBLIC
                                        15  483  485  487  489  502  507  508  593  594  665  669  672  744
                                       758  764  766  798  828  842  851  878  903

 16  0000H    1  MEMORYARG2           BYTE BASED(MEMORYARG2PTR)

 16  0004H    4  MEMORYARG2PTR        POINTER
                                        16  842  851

810  1C90H  133  MEMORYFILL           PROCEDURE STACK=003AH
                                       1055

834  1D15H  131  MEMORYMOVE           PROCEDURE STACK=003AH
                                       1054

474  13A9H  229  MEMORYTEST           PROCEDURE STACK=003AH
                                       721

                  MOVB                BUILTIN
                                       828  842  851  903  904  905  918

353  0067H    1  MSIGN                BYTE
                                       370  414

385  1005H       MULTLOOP             LABEL
                                       383

339  0008H    1  N                    BYTE PARAMETER AUTOMATIC
                                       340  354  355

272  0008H    1  N                    BYTE PARAMETER AUTOMATIC
                                       273  276  278  279

250  0008H    1  N                    BYTE PARAMETER AUTOMATIC
                                       251  252

174  0008H    1  N                    BYTE PARAMETER AUTOMATIC
                                       175  176

 75  0008H    1  N                    BYTE PARAMETER AUTOMATIC.
                                        76   77

 69  0008H    1  N                    BYTE PARAMETER AUTOMATIC
                                        70   71
```

```
463  0008H    2  N                    WORD PARAMETER AUTOMATIC
                                       464  465  467

451  0008H    2  N                    WORD PARAMETER AUTOMATIC
                                       452  453  455

 25  06B4H       NEXTCOMMAND          LABEL PUBLIC
                                       188  504  598  639  693  697  929  933  936  941 1040 1075

935  2008H   31  NMI                  PROCEDURE INTERRUPT(2) STACK=0002H
                                      1003

341  0000H    2  OFF                  WORD MEMBER(FLTADR)
                                       360

327  0000H    2  OFF                  WORD MEMBER(FLTADRS)
                                       329  330

320  0000H    2  OFF                  WORD MEMBER(FLTADRS)
                                       322  323

 17  0000H    2  OFF                  WORD MEMBER(BKPTR)
                                       912  927

 16  0000H    2  OFF                  WORD MEMBER(ARG2)

 15  0000H    2  OFF                  WORD MEMBER(ARG1)
                                       243  246  247  360  491  492  495  500  756  773  793  796  800  802
                                       805  830  873  876  880  882  885

986  0000H    2  OFF                  WORD MEMBER(VECTOR)
                                       988

343  0104H    4  ONE                  BYTE ARRAY(4) AT ABSOLUTE
                                       366  378

345  0024H    4  ONES                 BYTE ARRAY(4) DATA
                                       366

217  0057H    1  OPER                 BYTE
                                       218  226  233

521  002AH    2  OUTPORT              WORD
                                       526  532

                 OUTPUT               BUILTIN
                                        47   48   49   55   56   57  108  110  112  115  117  119  121  123
                                       129  134  136  334  444  446  448  456  458  613  614 1029 1030 1032
                                      1038 1039 1041

                 OUTWORD              BUILTIN
                                       323  330  441  515  532  539  548  565  682  683

939  2027H   46  OVERFLOWERROR        PROCEDURE INTERRUPT(4) STACK=002EH
                                      1005
```

```
  399  1159H        PROCESS                LABEL
                                           384

  268  0DCDH   18   PROMPT                 PROCEDURE STACK=0030H
                                           711  738  749  779  814  837  856  893

  436  12C3H   14   READCONSOLESWITCHES     PROCEDURE WORD PUBLIC STACK=0002H

   30  0000H        READDRIVER             PROCEDURE EXTERNAL(2) STACK=0000H
                                           1061

  607  1721H        REDO                   LABEL
                                           611

   36  0000H        RENAMEDRIVER           PROCEDURE EXTERNAL(5) STACK=0000H
                                           1064

  349  010CH    4   RES                    BYTE ARRAY(4) AT ABSOLUTE
                                           375  376  381  382

  217  000CH    2   SAVE                   WORD
                                           220  223  227  228  234

  352  005CH    8   SAVEDISPLAY            BYTE ARRAY(8)
                                           409  417  420

  342  0100H    4   SCRATCH                BYTE ARRAY(4) AT ABSOLUTE
                                           365  370  371  373  379  387  390  394  397  399

  708  005CH   13   SECCOMMAND             BYTE ARRAY(13) DATA
                                           715

  341  0002H    2   SEG                    WORD MEMBER(FLTADR)
                                           361

  327  0002H    2   SEG                    WORD MEMBER(FLTADRS)
                                           329

  320  0002H    2   SEG                    WORD MEMBER(FLTADRS)
                                           322

   17  0002H    2   SEG                    WORD MEMBER(BKPTR)
                                           910  925

   16  0002H    2   SEG                    WORD MEMBER(ARG2)

   15  0002H    2   SEG                    WORD MEMBER(ARG1)
                                           242  246  361  498

  986  0002H    2   SEG                    WORD MEMBER(VECTOR)
                                           987

  143  0A18H   25   SERIALCHARIN            PROCEDURE PUBLIC STACK=0002H
                                           182  312

  125  0964H   25   SERIALCHAROUT           PROCEDURE PUBLIC STACK=0004H
```

```
                                        160

                    SHL                 BUILTIN
                                        209  223  322  329  469

                    SHR                 BUILTIN
                                         61  166

       7            SIO8251CMND         LITERALLY
                                        123

       7            SIO8251MODE         LITERALLY
                                        121

       7            SIO8251RST          LITERALLY
                                        119

     250  0051H  42 SIOBLANKS           PROCEDURE PUBLIC STACK=0022H
                                        413  433  768  794  796  806

       5  0002H  18 SIOCMND             BYTE ARRAY(18) DATA
                                        996

     256  007BH  65 SIOCRLF             PROCEDURE PUBLIC STACK=0020H
                                        266  666  743

       7            SIODATAPORT         LITERALLY
                                        129  146  299

     776  1BB4H 220 SIODISPLAY          PROCEDURE STACK=003AH
                                        1053

     746  1AE3H 209 SIOEXAMMEM          PROCEDURE STACK=003AH
                                        1052

     240  0D24H  45 SIOGETADDR          PROCEDURE PUBLIC STACK=0032H
                                        358  480  506  592  740  751  781  816  839  844  858  900

     203  08DCH 126 SIOGETBYTE          PROCEDURE BYTE PUBLIC STACK=0028H
                                        763  824

     180  0B30H  60 SIOGETCHAR          PROCEDURE PUBLIC STACK=0020H
                                        210  224  237  245  305  307  479  512  525  529  559  563  585  591
                                        605  609  703  760  787  819  823  843  847  864  896  898  994

     216  0C5AH 202 SIOGETWORD          PROCEDURE WORD PUBLIC STACK=0028H
                                        243  247  513  526  530  560  564  606  704  788  820  848  865

     736  1AABH  56 SIOGO               PROCEDURE STACK=003AH
                                        1051

     196  0B88H  36 SIOHEX              PROCEDURE BYTE PUBLIC STACK=0004H
                                        209  223

     114  092BH  57 SIOINIT             PROCEDURE STACK=000CH
                                        1019
```

```
265  0D0CH   17  SIONEWLINE          PROCEDURE PUBLIC STACK=0020H
                                      269  285  460  471  477  496  503  510  523  527  557  561  574  583
                                      589  595  603  607  621  623  625  636  638  643  656  674  679  690
                                      694  701  755  792  804  908  913  917  923  928  945  950  956  961
                                      966  972  977  982  992 1026 1072 1074

164  0AA3H   52  SIOOUTBYTE          PROCEDURE PUBLIC STACK=0022H
                                      171  172  282  502  758  798

157  0A77H   44  SIOOUTCHAR          PROCEDURE PUBLIC STACK=0018H
                                      166  167  177  186  253  257  259  270  281  283  302  316  415  416
                                      417  418  420  425  426  428  429  430  431  499  544  581  671  757
                                      759  799  911  926

174  0AF9H   55  SIOOUTSTRING        PROCEDURE PUBLIC STACK=0026H
                                      434  461  472  476  478  497  501  508  511  522  524  528  537  545
                                      555  558  562  575  582  584  590  602  604  608  620  622  624  637
                                      642  655  668  678  691  695  700  702  710  737  748  769  778  813
                                      836  855  892  909  916  924  932  940  944  949  955  960  965  971
                                      976  981  993 1027 1028 1035 1036 1037 1044 1045 1073

169  0AD7H   34  SIOOUTWORD          PROCEDURE PUBLIC STACK=002CH
                                      498  500  576  692  696  756  793  805  910  912  925  927

  7              SIORXRDY            LITERALLY
                                      144  293

  7              SIOSTATPORT         LITERALLY
                                      115  117  119  121  123  127  144  293

  7              SIOTXRDY            LITERALLY
                                      127

190  0B6CH   76  SIOVALIDHEX         PROCEDURE BYTE PUBLIC STACK=0000H
                                      205  208  222

                 STACKPTR            BUILTIN
                                     1001 1049 1071

175  0000H    1  STR                 BYTE BASED(STRPTR) ARRAY(1)
                                      177

 70  0000H    1  STR                 BYTE BASED(STRPTR) ARRAY(1)
                                       72

174  000AH    4  STRPTR              POINTER PARAMETER AUTOMATIC
                                      175  177

 69  000AH    4  STRPTR              POINTER PARAMETER AUTOMATIC
                                       70   72

304  0E90H   24  SUPPRESSBLANKS      PROCEDURE PUBLIC STACK=0020H
                                      357  712  739  750  780  815  838  857

835  0049H    2  T                   WORD
                                      848  851
```

```
812  0072H    1  T                    BYTE
                                        824  828

747  0071H    1  T                    BYTE
                                        763  764  766

554  0034H    2  T                    WORD
                                        556  571

344  0108H    4  TEN                  BYTE ARRAY(4) AT ABSOLUTE
                                        368  388  395  403

348  002CH    4  TENS                 BYTE ARRAY(4) DATA
                                        368

618  006CH    1  TEST                 BYTE
                                        626  628  631

475  006AH    1  TEST                 BYTE
                                        481  482  486  490

                 TIME                 BUILTIN
                                         97  138

471  1390H      TIMEERROR              LABEL
                                        466

 20                TIMER0MODEWORD      LITERALLY
                                        444

 20                TIMER1MODEWORD      LITERALLY
                                        446

 20                TIMER2MODEWORD      LITERALLY
                                        448

 20                TIMERBASEPORT       LITERALLY
                                        455  467

 20                TIMERCONTROLPORT    LITERALLY
                                        444  446  448

460  1343H      TIMERERROR             LABEL
                                        454

443  12E4H    41  TIMERINIT            PROCEDURE STACK=000CH
                                       1021

451  130DH    79  TIMERLOAD            PROCEDURE PUBLIC STACK=0034H
                                        644  645  646

463  135CH    77  TIMERREAD            PROCEDURE WORD PUBLIC STACK=0032H

641  1002H    64  TIMERTEST            PROCEDURE STACK=003CH
                                        728
```

```
     4                  TRUE                    LITERALLY
                                                 193  221  275  481  482  514  531  538  547  626  628  754 1048

   372  104FH           TSTLP1                  LABEL
                                                 398

   378  1092H           TSTLP2                  LABEL
                                                 391

     1  04D0H   703     USAF86                  PROCEDURE STACK=0050H

   310  0EB3H    50     USERIOGETCHAR           PROCEDURE PUBLIC STACK=0020H
                                                 629

   464  0020H     2     VALUE                   WORD
                                                 469  470

   451  000AH     2     VALUE                   WORD PARAMETER AUTOMATIC
                                                 452  456  458

   439  0008H     2     VALUE                   WORD PARAMETER AUTOMATIC
                                                 440  441

   986  0000H     4     VECTOR                  STRUCTURE BASED(INTVECTPTR)
                                                 987  988

   217  000EH     2     W                       WORD
                                                 219  227  228  230

   169  0008H     2     W                       WORD PARAMETER AUTOMATIC
                                                 170  171  172

    64  0008H     2     W                       WORD PARAMETER AUTOMATIC
                                                  65   66   67

   554  002EH     2     W                       WORD
                                                 560  565  568

   439  12D1H    19     WRITECONSOLELEDS        PROCEDURE PUBLIC STACK=0004H

    32  0000H            WRITEDRIVER             PROCEDURE EXTERNAL(3) STACK=0000H
                                                 1062

   433  12A7H            ZEROEXIT                LABEL
                                                 363


MODULE INFORMATION:

     CODE AREA SIZE     = 2236H   8758D
     CONSTANT AREA SIZE = 0000H      0D
     VARIABLE AREA SIZE = 0074H    116D
     MAXIMUM STACK SIZE = 0050H     80D
     1453 LINES READ
     0 PROGRAM ERROR(S)
```

END OF PL/M-86 COMPILATION

II. DOS86 - THE DISK OPERATING SYSTEM

The disk operating system was designed to maintain to the degree possible compatibility with the Intel Microdevelopment System (MDS) and hence the ISIS disk operating system. The primary goal of the disk operating system was to allow the user to compile, link and locate his program using the MDS, convert to hexadecimal file format and then load directly to the 8086 from the flexible disk media. This type of system was necessary as the user's MDS is physically separted from the point of operation of the USAFA/8086. Further, this operating system provides an ideal opportunity to venture into the compiler, editor, etc., software areas from the USAFA/8086.

The following primary control characters are recognized by DOS86:

- R: Read a file directly into memory
- W: Write a file directly from memory
- D: Display a listing to the console of the files on the diskette
- E: Erase a given file
- I: Input a hexadecimal file into memory (primary download)
- K: Copy a file from one drive to the other
- N: Namechange of an existing file
- A: Attribute change of an existing file

Each of these capabilities relies on the existance of a set of utility routines. Part of our utility routines are grouped in a submodular called DATA TRANSFER ROUTINES. Figure 8 shows these procedures.

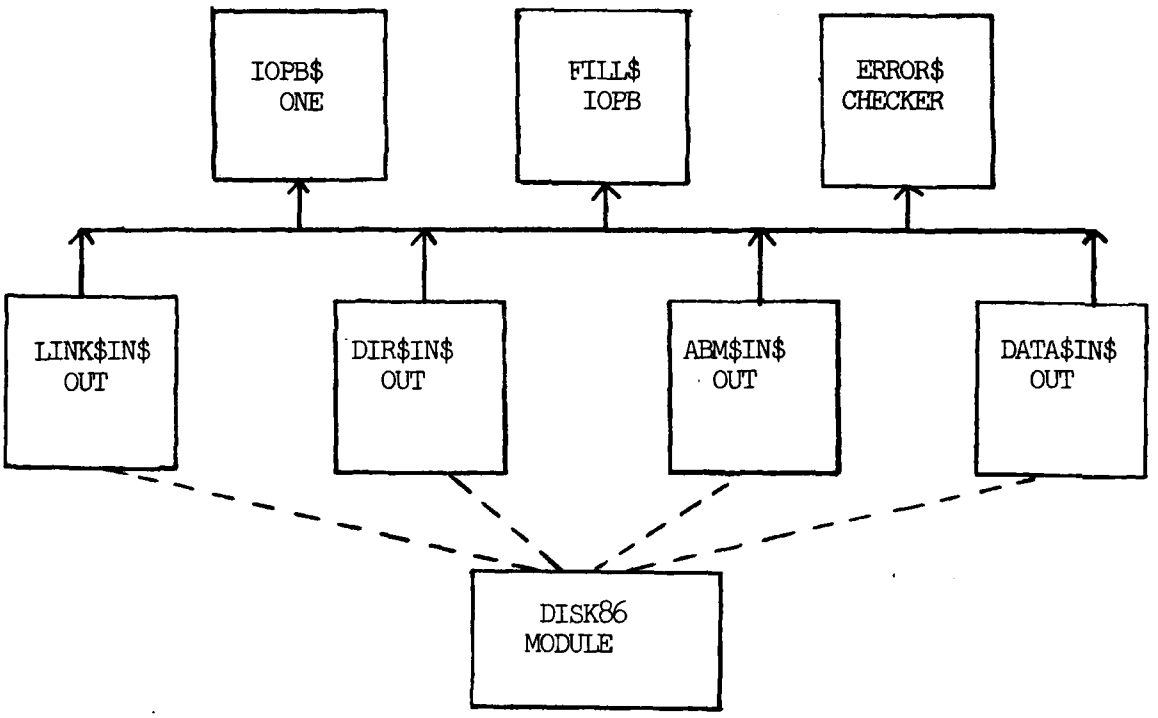


FIGURE 8. DATA TRANSFER UTILITIES

In addition to these utilities a group of relatively unrelated procedures are necessary to support operations. Six such procedures are included in a submodule called SUPPORT ROUTINES. Table 3 shows the functions of these routines.

TABLE 3. SUPPORT PROCEDURES

ROUTINE	FUNCTION
ABM$ZERO	Makes a given sector on a given track available for further use.
FREE$SECTOR	Identifies an available sector on a track and alters the ABM.
MAKE$LINK	Given the number of required sectors, this procedure creates an appropriate linkage map.
AVAILABLE$ SECTORS	Determines the number of free sectors remaining on the diskette.
FIND$FILE	Determines if a named file exists and locates it.
AVAILDIR	Determines the availability for a directory
ENTRY	entry to be made.

The command routines were written on a separate submodule from the driver routines to allow for the possibility of direct user access to the disk subsystem through this software. The easiest approach to description here seems to be to chart the interconnections of each command routine with those support procedures it uses. The RENAME procedure is shown in Figure 9.

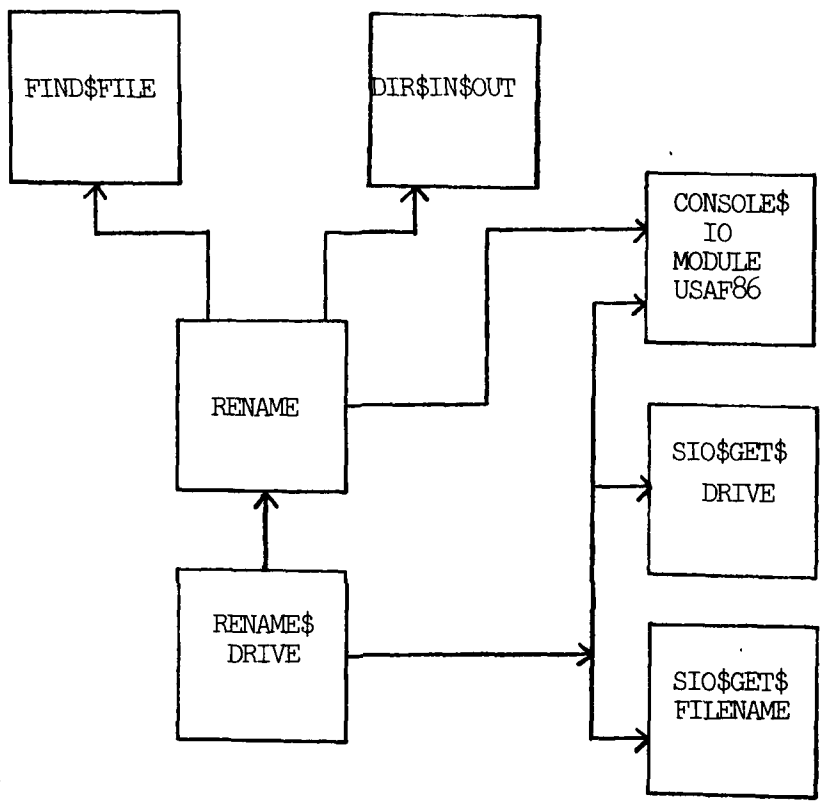


FIGURE 9. RENAME PROCEDURE RELATIONS

The attribute change routine is not much different and is diagrammed in Figure 10.

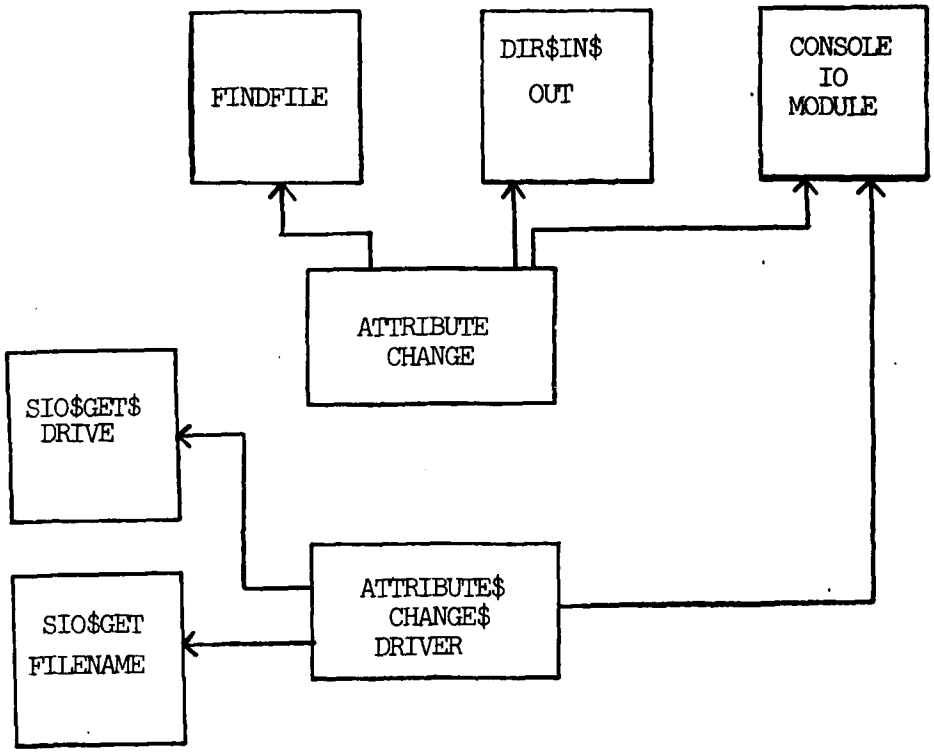


FIGURE 10. ATTRIBUTE CHANGE PROCEDURE RELATIONS

The erase command is diagrammed in Figure 11.

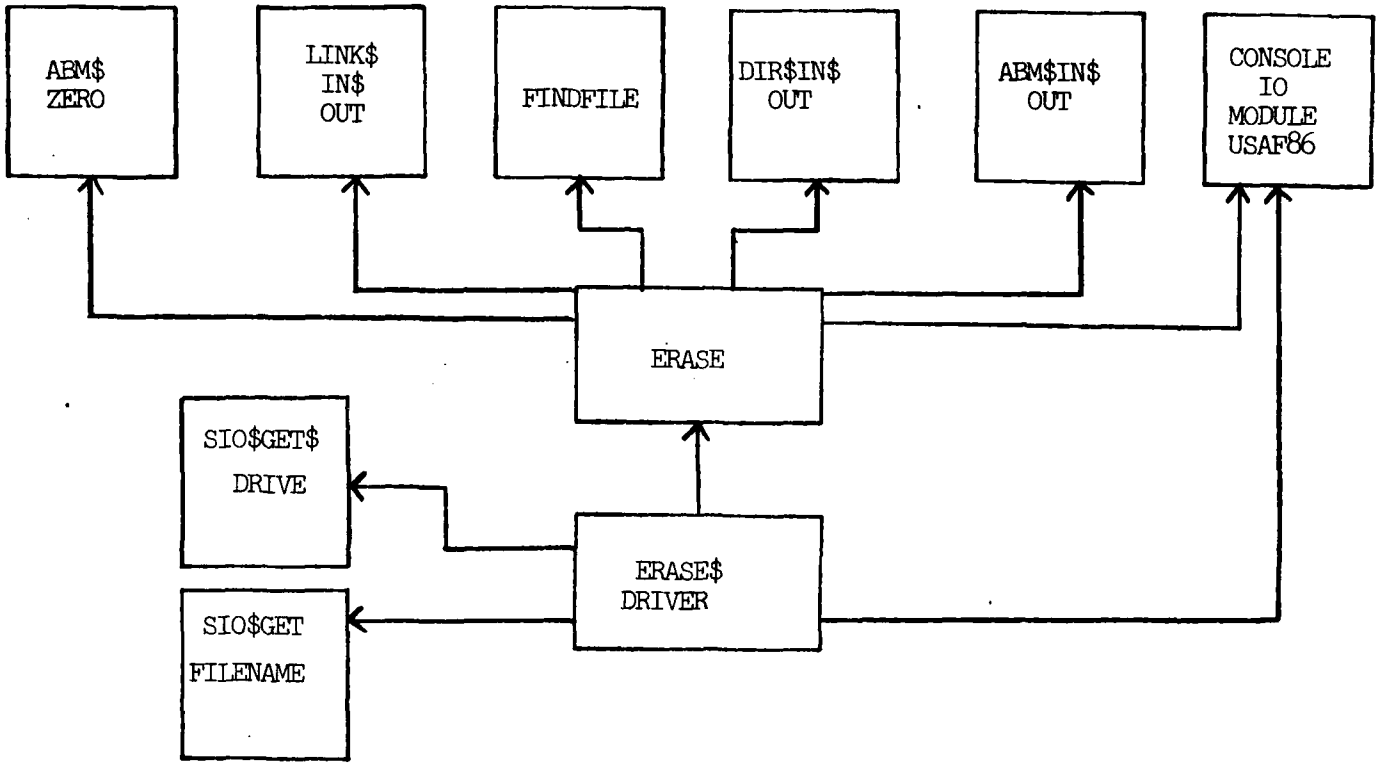


FIGURE 11. ERASE PROCEDURE RELATIONS

The erase procedure actually does not erase the data, rather it changes the flag in the directory to indicate the file is non-active and makes the sectors available by altering the allocation bit map. An erased file can actually be recovered if no additional files have been created since the erasure.

The copy procedure (spelled KOPY to avoid confusion with the use of C for the clear breakpoint function in USAF86) is reasonably complicated since it uses files on each of the two disk drives. The kopy routine was designed to allow use with a disk formatting routine which is not included with DOS86. The kopy procedure is detailed in Figure 12.

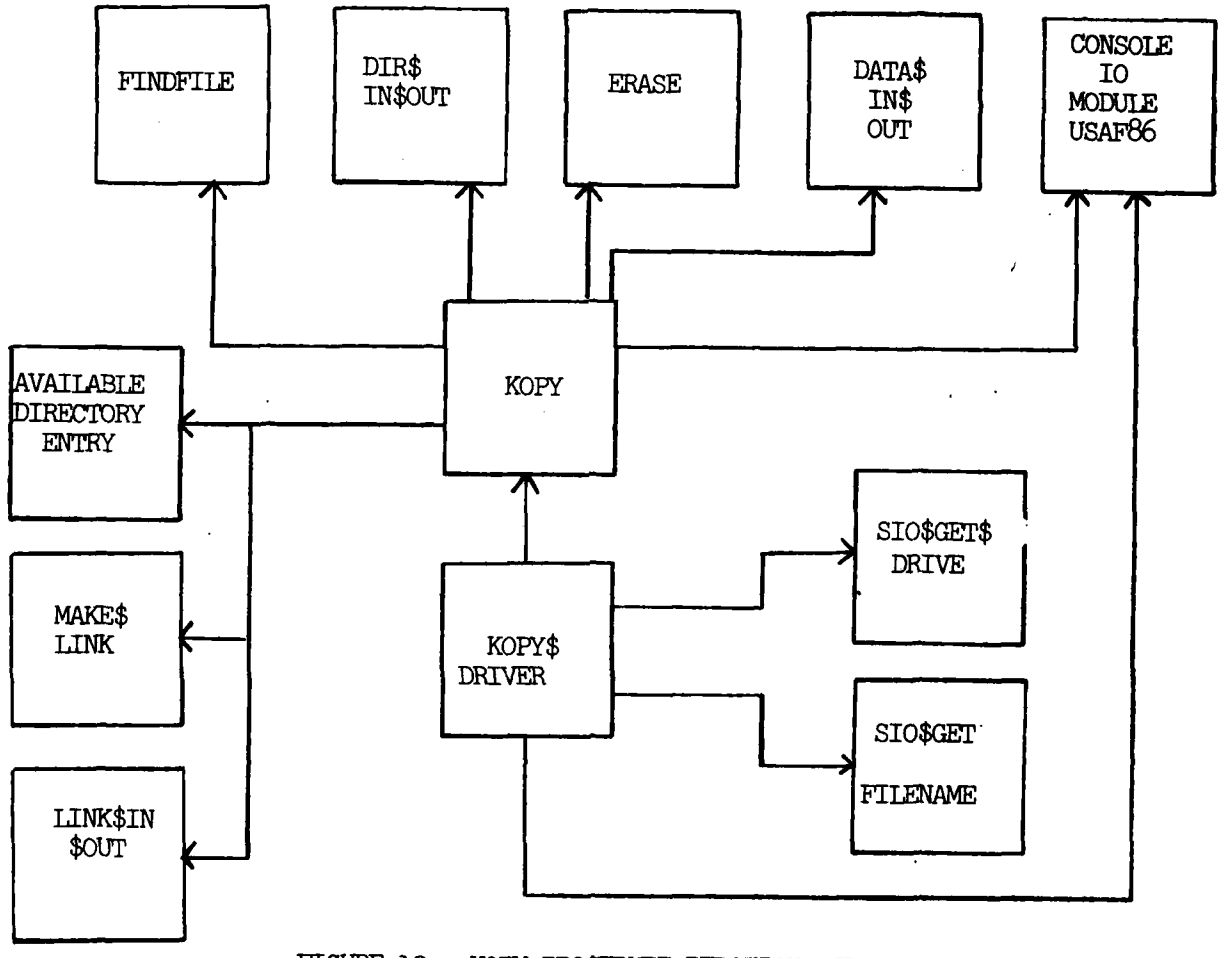


FIGURE 12. KOPY PROCEDURE RELATIONSHIPS

The read routine is implemented very easily using the previously mentioned support routines. These relationships are shown in Figure 13.

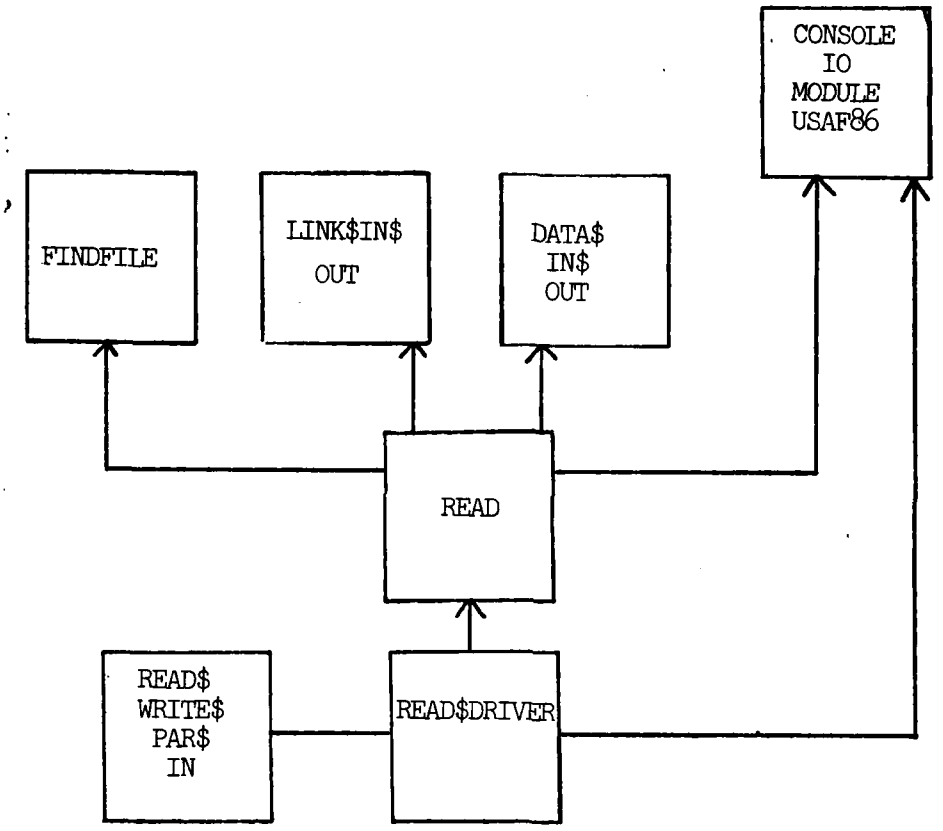


FIGURE 13. READ PROCEDURE RELATIONSHIPS

The READ$WRITE$PAR$IN routine used in the Read routine is merely a convenient procedure used to detail both the drive, name, and optionally the start and stop address. This routine communicates directly only with the console I/O module of USAF86. Default addresses are supplied for both the start address (01000H) and the stop address (07FFFH). This default area consumes almost all of the user available memory.

The WRITE command is quite similar to the structure of the READ command. It is necessary to emphasize that these two routines deal strictly with a "core image" file. No translation of the data or data records takes place in either of these routines. The WRITE command can be diagrammed as seen in Figure 14.

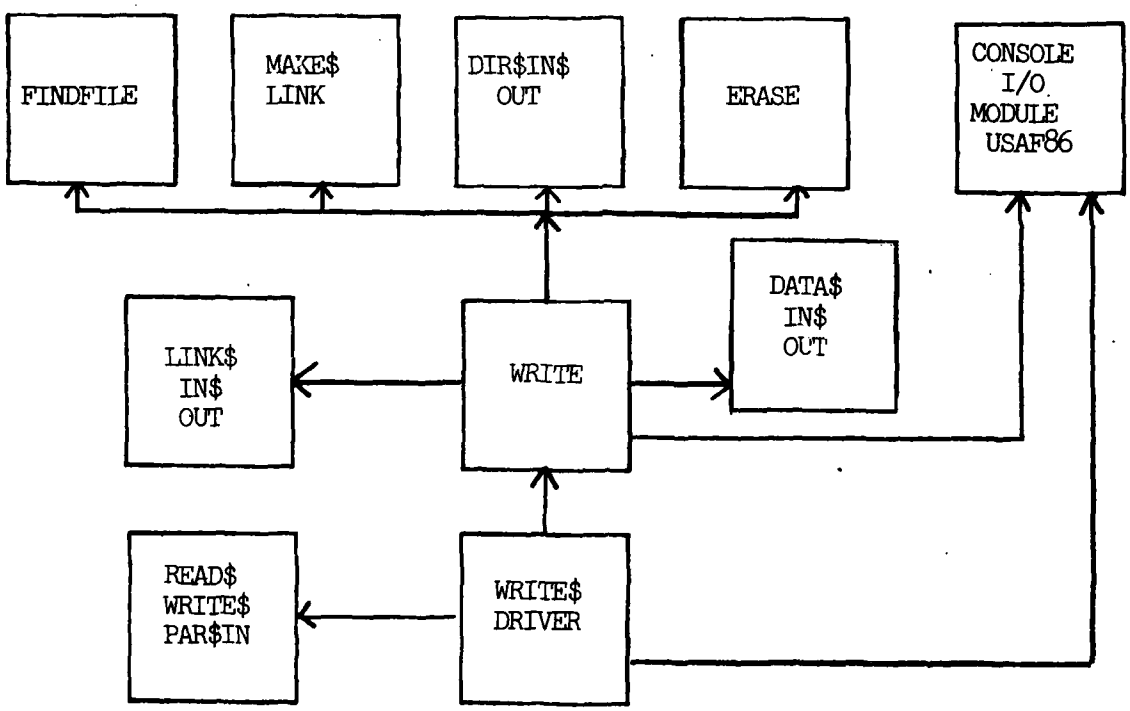


FIGURE 14. WRITE COMMAND RELATIONSHIPS

The DIRECTORY command allows the user to display the files contained on any diskette to the console. An indication is given to the user of the length of each file and the total utilization of the diskette. The DIRECTORY command is detailed in Figure 15.

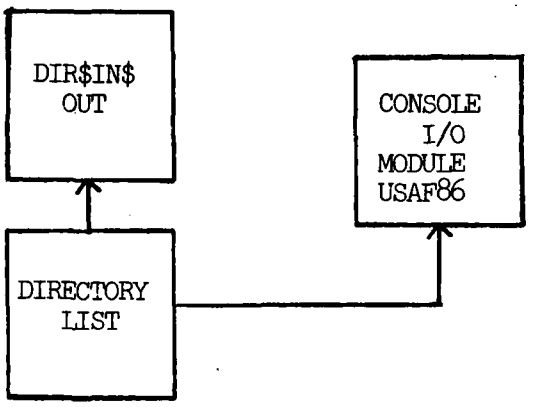


FIGURE 15. DIRECTORY COMMAND RELATIONSHIPS

The INPUTHEXFILE (called INITIALIZE in the listing) routine allows downloading into the USAFA/8086 any file created through the MDS-311 Utility called OH86. This routine functions similarly to the READ routine except that this routine interprets the code according to the file format indicated in the MCS-86 Absolute Object File Formats Technical Specification. The diagram of the INPUTHEXFILE routine is shown in Figure 16.

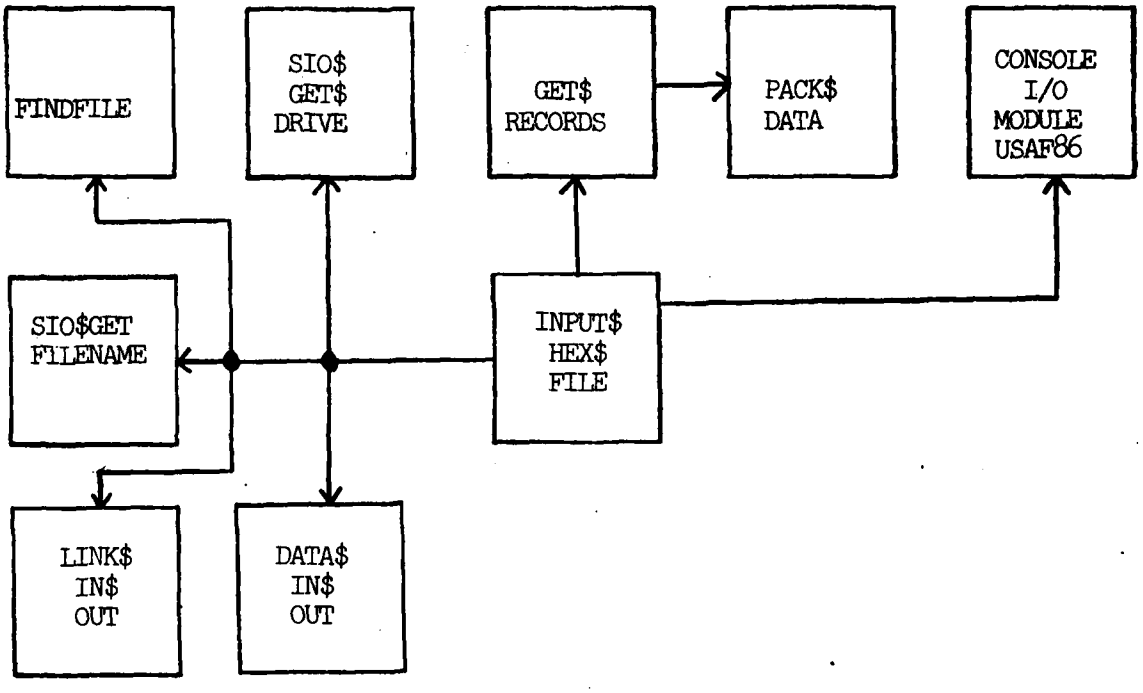


FIGURE 16. INPUTHEXFILE RELATIONSHIPS

Table 4 indicates the memory locations absolutely used by the DOS86 module.

TABLE 4. DOS86 DEDICATED MEMORY LOCATIONS

LOCATION	FUNCTION
00400-0047F	Link Map Buffer #1
00480-004FF	Link Map Buffer #2
00500-0057F	Directory Buffer #1
00580-005FF	Directory Buffer #2
00600-0067F	DATA Buffer
00680-0077F	Allocation Bit Map Buffer
00780-0078A	Input Output Parameter Block Buffer

This would indicate that if pressed, the user programming might actually start as low as 0078B .

The following pages are the actual PLM86 listing of DOS86.

```
ISIS-II PL/M-86 V1.1 COMPILATION OF MODULE DOS
NO OBJECT MODULE REQUESTED
COMPILER INVOKED BY:  PLM86 :F1:FITZ.SRC NOOBJECT DATE(30 APR 80) XREF


            $TITLE('8086 DISK MANAGEMENT SYSTEM') LARGE
  1         DOS:DO;
              /* THE FOLLOWING SOFTWARE IS THE DISK MANAGEMENT SYSTEM TO
              THE USAFA/8086 MICROCOMPUTER.  THE FUNCTIONS THAT  IT WILL
              PERFORM INCLUDE "READ", "WRITE", "KOPY", "NAMECHANGER" (RENAME),
              "ERASE", "ATTRIBUTE CHANGE", "INITIALIZE", AND "DIRECTORY".

              THE SOFTWARE IS MODULAR IN NATURE CONSISTING OF THE FOLLOWING
              SEQUENTIAL BLOCKS:
                 1) EXTERNAL ROUTINES
                 2) GLOBAL DECLARATIONS
                 3) DATA TRANSFER ROUTINES
                 4) SUPPORT ROUTINES
                 5) COMMAND ROUTINES
                 6) DRIVER ROUTINES

              THIS SOFTWARE WAS WRITTEN AS PART OF AN INDEPENDENT RESEARCH
              PROJECT, EE499, BY CIC GLENN D. ROSENBERGER AND CIC PHILIP
              B. FITZJARRELL DURING THE FALL SEMESTER OF 1979. */
```

```
                $SUBTITLE('EXTERNAL ROUTINES')
                    /* THE FOLLOWING PACKAGE INCLUDES EXTERNAL SUBROUTINES
                    THAT ARE USED TO ENTER PARAMETERS FROM THE KEYBOARD
                    AND DISPLAY RESULTS ON THE CRT */
                $INCLUDE (:F0:EXTRN.PLM)
 2   1  =    DELAY:
        =        PROCEDURE(C) EXTERNAL;
 3   2  =        DECLARE C BYTE;
 4   2  =        END;
 5   1  =    DELAY$LONG:
        =        PROCEDURE EXTERNAL;
 6   2  =        END;
 7   1  =    SIO$OUT$CHAR:
        =        PROCEDURE(C) EXTERNAL;
 8   2  =        DECLARE C BYTE;
 9   2  =        END;
10   1  =    SIO$OUT$BYTE:
        =        PROCEDURE(B) EXTERNAL;
11   2  =        DECLARE B BYTE;
12   2  =        END;
13   1  =    SIO$OUT$WORD:
        =        PROCEDURE(W) EXTERNAL;
14   2  =        DECLARE W WORD;
15   2  =        END;
16   1  =    SIO$OUT$STRING:
        =        PROCEDURE(STR$PTR,N) EXTERNAL;
17   2  =        DECLARE STR$PTR POINTER, N BYTE;
18   2  =        END;
19   1  =    SIO$GET$CHAR:
        =        PROCEDURE EXTERNAL;
20   2  =        END;
21   1  =    SIO$VALID$HEX:
        =        PROCEDURE(H) EXTERNAL;
22   2  =        DECLARE H BYTE;
23   2  =        END;
24   1  =    SIO$HEX:
        =        PROCEDURE(C) BYTE EXTERNAL;
25   2  =        DECLARE C BYTE;
26   2  =        END;
27   1  =    SIO$GET$BYTE:
        =        PROCEDURE BYTE EXTERNAL;
28   2  =        END;
29   1  =    SIO$GET$WORD:
        =        PROCEDURE WORD EXTERNAL;
30   2  =        END;
31   1  =    SIO$GET$ADDR:
        =        PROCEDURE(DEFAULT$BASE) EXTERNAL;
32   2  =        DECLARE DEFAULT$BASE WORD;
33   2  =        END;
34   1  =    SIO$BLANKS:
        =        PROCEDURE(N) EXTERNAL;
35   2  =        DECLARE N BYTE;
36   2  =        END;
37   1  =    SIO$CRLF:
        =        PROCEDURE EXTERNAL;
38   2  =        END;
```

```
39  1  =  SIO$NEWLINE
       =      PROCEDURE EXTERNAL;
40  2  =      END;
41  1  =  SUPPRESS$BLANKS:
       =      PROCEDURE EXTERNAL;
42  2  =      END;
43  1  =  USER$IO$GET$CHAR:
       =      PROCEDURE EXTERNAL;
44  2  =      END;
45  1  =  DECLARE MEMORY$ARG1$PTR POINTER EXTERNAL, ARG1 STRUCTURE
       =      (OFF WORD, SEG WORD) AT (@MEMORY$ARG1$PTR),
       =      MEMORY$ARG1 BASED MEMORY$ARG1$PTR BYTE;
46  1  =  DECLARE (ERROR,NEXTCOMMAND) LABEL EXTERNAL;
47  1  =  DECLARE CHAR BYTE EXTERNAL;
48  1  =  DECLARE ASCR LITERALLY '0DH';
49  1  =  DECLARE ASBL LITERALLY '020H';
```

```
$SUBTITLE('GLOBAL DECLARATIONS')

    /* THE FOLLOWING BLOCKS OF RAM ARE USED AS BUFFERS FOR THE
    DATA TRANSFER BETWEEN THE MICROCOMPUTER AND THE DISK:
        00400H - 0047FH: LINKAGE MAP SECTOR BUFFER #1
        00480H - 004FFH: LINKAGE MAP SECTOR BUFFER #2
        00500H - 0057FH: DIRECTORY SECTOR BUFFER #1
        00580H - 005FFH: DIRECTORY SECTOR BUFFER #2
        00600H - 0067FH: DATA BUFFER
        00680H - 0077FH: ALLOCATION BIT MAP BUFFER */

50  1   DECLARE DIRECTORYA(8) STRUCTURE(FLAG BYTE,FNAME(6) BYTE,
        NMEXT(3) BYTE,ATTR BYTE,LAST$SECT$BYTE BYTE,NUM$SECT WORD,
        FST$LNK$SECT BYTE,FST$LNK$TRK BYTE) AT (00500H);

51  1   DECLARE DIRECTORYB(8) STRUCTURE(FLAG BYTE,FNAME(6) BYTE,
        NMEXT(3) BYTE,ATTR BYTE,LAST$SECT$BYTE BYTE,NUM$SECT WORD,
        FST$LNK$SECT BYTE,FST$LNK$TRK BYTE) AT (00580H);

52  1   DECLARE LINK1(64) STRUCTURE(SECT BYTE,TRK BYTE) AT (00400H);

53  1   DECLARE LINK2(64) STRUCTURE(SECT BYTE,TRK BYTE) AT (00480H);

54  1   DECLARE ABM(256) BYTE AT (00680H);

55  1   DECLARE DDATA(128) BYTE AT (00600H);

56  1   DECLARE TRUE LITERALLY '0FFH';
57  1   DECLARE (FILE$SECTORA,FILE$IDENTA,FILE$SECTORB,FILE$IDENTB,
        FST$LNK$SECT,FST$LNK$TRK,SECTOR$NUMBER,TRACK$NUMBER,
        DRIVE1,DRIVE2) BYTE, (STADD,FNADD) WORD;

58  1   DECLARE FILENAME1(6) BYTE,FILENAME2(6) BYTE,FILEEXT1(3) BYTE,
        FILEEXT2(3) BYTE;

59  1   DECLARE IOPB WORD EXTERNAL;

60  1   DECLARE (FTERR,DERR) BYTE EXTERNAL;

61  1   DECLARE IOBP(10) BYTE AT (00780H);

62  1   DECLARE LAST$ABM$BYTE WORD ;

63  1   DECLARE ABC(*) BYTE DATA ('ABCDEFGHIJKLMNOPQRSTUVWXYZ');

64  1   DECLARE ALPHANUM(*) BYTE DATA ('ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789');

65  1   DECLARE DIRA$BUF$ADRS   WORD    DATA(00500H),
        DIRB$BUF$ADRS   WORD    DATA(00580H),
        LINK1$BUF$ADRS  WORD    DATA(00400H),
        LINK2$BUF$ADRS  WORD    DATA(00480H),
        ABM$BUF$ADRS    WORD    DATA(00680H),
        DDATA$BUF$ADRS  WORD    DATA(00600H);
66  1   DECLARE DATA$POINTER POINTER,
        DPNTR STRUCTURE(OFF WORD, SEG WORD) AT (@DATA$POINTER);
67  1   DECLARE TARGET$POINTER POINTER,
```

```
                TPNTR STRUCTURE(OFF WORD, SEG WORD) AT (@TARGET$POINTER);
 68   1        DECLARE START$POINTER POINTER,
                IST STRUCTURE (IP WORD, CS WORD) AT (@START$POINTER);
```

```
                $SUBTITLE('DATA TRANSFER ROUTINES')

                   /* THE FOLLOWING ROUTINE IS CALLED TO TRANSFER DATA
                   TO AND FROM THE DISK. */
     69   1      DISK:
                  PROCEDURE EXTERNAL;
     70   2      END;

                   /* THE FOLLOWING ROUTINE FILLS THE INPUT/OUTPUT
                   PARAMATER BLOCK SO THAT THE PROGRAM CAN 'TALK' TO
                   THE DISK. */
     71   1      FILL$IOPB:
                  PROCEDURE (X0,X1,X2,X3,X4,X5,X6,X7,X8,X9);
     72   2          DECLARE (X0,X1,X2,X3,X4,X5,X6,X7,X8,X9) BYTE;
     73   2          IOBP(0)=X0;
     74   2          IOBP(1)=X1;
     75   2          IOBP(2)=X2;
     76   2          IOBP(3)=X3;
     77   2          IOBP(4)=X4;
     78   2          IOBP(5)=X5;
     79   2          IOBP(6)=X6;
     80   2          IOBP(7)=X7;
     81   2          IOBP(8)=X8;
     82   2          IOBP(9)=X9;
     83   2          IOPB=0780H;  /*PUBLIC BYTE SPECIFYING LOCATION OF IOPB*/
     84   2      END FILL$IOPB;

                   /* THE FOLLOWING ROUTINE IS CALLED TO SEE IF THE DISK
                   FAILS TO OPERATE PROPERLY.  IT OUTPUTS AN ERROR
                   MESSAGE AND RESETS THE SYSTEM FOR THE NEXT COMMAND. */
     85   1      ERROR$CHECKER:
                  PROCEDURE;
     86   2          FTERR=FTERR+1-1;/*DUMMY FOR COMPILER*/
     87   2          IF (FTERR <> 0) OR (DERR <> 0) THEN DO;
     89   3              CALL SIO$OUT$STRING(@('FATAL ERROR, DISK RELATED'),25);
     90   3              GO TO NEXT$COMMAND;
     91   3              END;
     92   2          ;
     93   2      END ERROR$CHECKER;

                   /* THE FOLLOWING PROCEDURE IS USED TO ESTABLISH IOPB(1),
                   WHICH TELLS THE DISK WHICH DISK DRIVE IS BEING USED AND
                   WHETHER THE COMMAND IS A READ (INOUT=0) OR A WRITE (INOUT=1). */
     94   1      IOPB$ONE:
                  PROCEDURE (INOUT,DRIVE) BYTE;
     95   2          DECLARE (INOUT,DRIVE,X1) BYTE;
     96   2          INOUT=INOUT+1-1;/*DUMMY STATEMENT*/
     97   2              IF INOUT=0 THEN X1=04H;          /* READ */
     99   2              ELSE X1=06H;                  /* WRITE */
    100   2          X1=X1 OR SHL(DRIVE,4);
    101   2          RETURN X1;
    102   2      END IOPB$ONE;

                   /* THE FOLLOWING PROCEDURE IS USED TO READ OR WRITE A DIRECTORY
                   SECTOR INTO OR FROM ONE OF THE TWO DIRECTORY BUFFFERS IN SYSTEM
                   RAM.  'INOUT' SPECIFIES READ DISK (INOUT=0) OR WRITE TO DISK (INOUT=1).
```

```
                    'DRIVE' SPECIFIES WHICH DISK DRIVE IS TO BE EMPLOYED. 'SECTOR'
                    SPECIFIES WHICH OF THE 25 DIRECTORY SECTORS IS TO BE READ OR
                    WROTE. FINALLY, 'IDENT' SPECIFIES WHETHER TO USE DIRECTORYA (0)
                    OR DIRECTORYB(1).                                          */
103   1         DIR$IN$OUT:
                PROCEDURE (INOUT,DRIVE,SECTOR,IDENT);
104   2             DECLARE (INOUT,DRIVE,SECTOR,IDENT,P1,P5,P6) BYTE;
105   2             P1=IOPB$ONE(INOUT,DRIVE);   /* SPECIFY READ/WRITE AND DRIVE */
106   2             IDENT=IDENT+1-1;/* DUMMY STATEMENT*/
107   2             IF IDENT=0 THEN
108   2                DO;                 /* SPECIFY BUFFER LOCATION */
109   3                   P5=LOW(DIRA$BUF$ADRS);
110   3                   P6=HIGH(DIRA$BUF$ADRS);
111   3                END;
                     ELSE
112   2                DO;
113   3                   P5=LOW(DIRB$BUF$ADRS);
114   3                   P6=HIGH(DIRB$BUF$ADRS);
115   3                END;
116   2             CALL FILL$IOPB(00H,P1,01,01,SECTOR,P5,P6,0,0,0);
117   2             CALL DISK;
118   2             CALL ERROR$CHECKER;
119   2         END DIR$IN$OUT;

                    /* THE FOLLOWING PROCEDURE IS USED TO TRANSFER THE ALLOCATION
                    BIT MAP OF THE DISK IN THE DRIVE SPECIFIED BY 'DRIVE' WITH
                    INOUT SPECIFYING A READ FROM DISK TO THE ABM BUFFER (INOUT=0),
                    OR A WRITE TO DISK FROM THE ABM BUFFER (INOUT=1)            */
120   1         ABM$IN$OUT:
                PROCEDURE (INOUT,DRIVE);
121   2             DECLARE (INOUT,DRIVE,P1,P5,P6) BYTE;
122   2             P1=IOPB$ONE(INOUT,DRIVE);  /* SPECIFY READ/WRITE AND DRIVE */
123   2             P5=LOW(ABM$BUF$ADRS);
124   2             P6=HIGH(ABM$BUF$ADRS);     /* SPECIFY BUFFER LOCATION */
125   2             CALL FILL$IOPB(00H,P1,01,02,02,P5,P6,0,0,0);
126   2             CALL DISK;
127   2             CALL ERROR$CHECKER;
128   2             CALL FILL$IOPB(00H,P1,01,02,03,00,07,00,00,00);
129   2             CALL DISK;
130   2             CALL ERROR$CHECKER;
131   2         END ABM$IN$OUT;

                    /*THE FOLLOWING PROCEDURE IS USED TO READ (INOUT=0) OR WRITE
                    (INOUT=1) A SECTOR OF A LINKAGE MAP TO OR FROM THE DISK
                    SPECIFIED BY 'DRIVE'. 'TRACK' AND 'SECTOR' IDENTIFY THE
                    LOCATION OF THE LINKAGE MAP SECTOR ON THE DISK DRIVE, AND
                    'IDENT' IS USED TO IDENTIFY WHICH OF THE TWO LINKAGE MAP
                    BUFFERS (IDENT=0 IS LINK1, IDENT=1 IS LINK2) THAT IS TO BE USED.*/
132   1          LINK$IN$OUT:
                PROCEDURE (INOUT,DRIVE,TRACK,SECTOR,IDENT);
133   2             DECLARE (INOUT,DRIVE,TRACK,SECTOR,IDENT,P1,P5,P6) BYTE;
134   2             P1=IOPB$ONE(INOUT,DRIVE);  /* SPECIFY READ/WRITE AND DRIVE */
135   2             IDENT=IDENT+1-1;/* DUMMY COMPILER STATEMENT*/
136   2             IF IDENT=0 THEN
137   2                DO;                 /* SPECIFY BUFFER LOCATION */
138   3                   P5=LOW(LINK1$BUF$ADRS);
```

```
139  3              P6=HIGH(LINK1$BUF$ADRS);
140  3            END;
                 ELSE
141  2            DO;
142  3              P5=LOW(LINK2$BUF$ADRS);
143  3              P6=HIGH(LINK2$BUF$ADRS);
144  3            END;
145  2          CALL FILL$IOPB(00H,P1,01,TRACK,SECTOR,P5,P6,0,0,0);
146  2          CALL DISK;
147  2          CALL ERROR$CHECKER;
148  2      END LINK$IN$OUT;

          /* THE FOLLOWING PROCEDURE IS USED TO TRANSFER GENERAL
          DATA TO OR FROM THE DISK TRACK AND SECTOR SPECIFIED BY THE
          FORMAL PARAMETERS.  INOUT IS USED TO DETERMINE WHETHER IT IS
          A READ (INOUT=0) OR WRITE (INOUT=1).  IF DATAADDRESS=0 THEN THE DATA
          TRANSFERED COMES FROM OR GOES TO THE DATA BUFFER.  IF DATAADRESS<>0,
          THEN THE DATA IS TRANSFERED TO THE LOCATION SPECIFIED BY DATAADRESS. */
149  1      DATA$IN$OUT:
          PROCEDURE (INOUT,DRIVE,TRACK,SECTOR,DATAADRESS);
150  2          DECLARE (INOUT,DRIVE,TRACK,SECTOR,P1,P5,P6) BYTE;
151  2          DECLARE DATAADRESS WORD;
152  2          P1=IOPB$ONE(INOUT,DRIVE);  /* SPECIFY READ/WRITE AND DRIVE */
153  2          DATAADRESS=DATAADRESS+1-1;/*DUMMY COMPILER STATEMENT*/
154  2          IF DATAADRESS=0 THEN
155  2              DO;                /* SPECIFY BUFFER LOCATION */
156  3                P5=LOW(DDATA$BUF$ADRS);
157  3                P6=HIGH(DDATA$BUF$ADRS);
158  3              END;
                 ELSE
159  2              DO;
160  3                P5=LOW(DATAADRESS);
161  3                P6=HIGH(DATAADRESS);
162  3              END;
163  2          CALL FILL$IOPB(00H,P1,01,TRACK,SECTOR,P5,P6,0,0,0);
164  2          CALL DISK;
165  2          CALL ERROR$CHECKER;
166  2      END DATA$IN$OUT;
```

```
                $SUBTITLE('SUPPORT ROUTINES')

                   /* THE FOLLOWING PROCEDURE IS USED TO WRITE A ZERO TO THE
                   ALLOCATION BIT MAP LOCATION CORRESPONDING TO THE SECTOR
                   AND TRACK CORRESPONDING TO THE FORMAL PARAMETERS, THUS
                   MAKING THE SECTOR AVAILABLE                              */
  167   1       ABM$ZERO:
                PROCEDURE (TRACK,SECTOR);
  168   2          DECLARE (TRACK,SECTOR,ABMBYTE,MASK) BYTE;
  169   2          DECLARE (ABMNUM,SHIFTS,I) WORD;
  170   2          ABMNUM=(DOUBLE(TRACK)*26)+DOUBLE(SECTOR)-1;
  171   2          ABMBYTE=(ABMNUM/8);      /* DETERMINE PROPER BYTE IN BUFFER */
  172   2          SHIFTS=(ABMNUM MOD(8));  /* DETERMINE PROPER BIT IN BYTE */
  173   2          MASK=7FH;
  174   2          DO I=1 TO SHIFTS;              /* DEVELOP MASK */
  175   3             MASK=ROR(MASK,1);
  176   3          END;
  177   2          ABM(ABMBYTE)=(ABM(ABMBYTE) AND MASK);
  178   2       END ABM$ZERO;

                   /*THE FOLLOWING PROCEDURE IS CALLED IF THE OPERATOR
                   MAKES AN INVALID COMMAND ENTRY.                          */
  179   1       COMMAND$ERROR:
                PROCEDURE;
  180   2          IF CHAR<>ASCR THEN          /* WAIT UNTIL USER ENTERS A 'RETURN' */
  181   2          DO WHILE CHAR<>ASCR;        /* BEFORE AN ERROR MESSAGE IS PRINTED. */
  182   3             CALL SIO$GET$CHAR;
  183   3          END;
  184   2          GO TO ERROR;
  185   2       END COMMAND$ERROR;

                   /*THE FOLLOWING PROCEDURE FINDS A FREE SECTOR AND SETS THE
                   GLOBALS SECTOR$NUMBER AND TRACK$NUMBER TO IDENTIFY IT.
                   THE ALLOCATION BIT MAP OF THE PROPER DISK DRIVE SHOULD BE
                   ALREADY STORED IN THE BUFFER, AND THIS PROCEDURE CHANGES
                   THE BUFFER BUT DOES NOT REWRITE THE ABM TO THE DISK.      */
  186   1       FREE$SECTOR:
                PROCEDURE;
  187   2          DECLARE MASK BYTE;
  188   2          DECLARE (BYTECOUNT,SHIFTCOUNT) WORD;
  189   2          DECLARE (SHIFTS,ABM$BYTE) WORD;
  190   2          DECLARE FOUND LABEL;
  191   2          DO ABM$BYTE=0 TO 250;
  192   3             BYTECOUNT=ABMBYTE;
  193   3             MASK=080H;
  194   3             DO SHIFTS=1 TO 8;
  195   4                SHIFTCOUNT=SHIFTS;
  196   4                IF (ABM(ABM$BYTE) AND MASK)=0 THEN GO TO FOUND;
  198   4                MASK=SHR(MASK,1);
  199   4             END;
  200   3          END;
  201   2          FOUND:
                      /* SPECIFY SECTOR LOCATION */
                   SECTOR$NUMBER=LOW((BYTECOUNT*8+SHIFTCOUNT)MOD(26))+1;
                      /* SPECIFY TRACK LOCATION */
  202   2          TRACK$NUMBER=LOW((BYTECOUNT*8+SHIFTCOUNT)/26);
```

```
203  2          LAST$ABM$BYTE=BYTECOUNT;
204  2          ABM(BYTECOUNT)=ABM(BYTECOUNT) OR MASK;
205  2      END FREE$SECTOR;

              /*THE FOLLOWING PROCEDURE MAKES A LINK MAP ON THE DISK
              DRIVE SPECIFIED BY 'DRIVE' FOR 'NUMSECTOR' NUMBER OF SECTORS.*/
206  1      MAKE$LINK:
              PROCEDURE (NUMSECTOR,DRIVE);
207  2          DECLARE NUMSECTOR WORD;
208  2          DECLARE (DRIVE,LAST$LINK$SECT,LAST$LINK$TRK,LINK$SECTOR) BYTE;
209  2          DECLARE LINK$TRACK BYTE;
210  2          DECLARE (IND,INDD,LND) WORD;
211  2          DECLARE LAST$LINK LABEL;
212  2          CALL ABM$IN$OUT(0,DRIVE);              /* READ ABM */
213  2          LAST$LINK$SECT,LAST$LINK$TRK=0;
214  2          CALL FREE$SECTOR;
                /* FIRST LINKAGE MAP SECTOR */
215  2          FST$LNK$SECT,LINK$SECTOR=SECTOR$NUMBER;
216  2          FST$LNK$TRK,LINK$TRACK=TRACK$NUMBER;
217  2          DO WHILE TRUE;
                /* LOCATION OF PREVIOUS LINKAGE MAP SECTOR */
218  3              LINK2(0).TRK=LAST$LINK$TRK;
219  3              LINK2(0).SECT=LAST$LINK$SECT;
220  3              DO IND=2 TO 63;
221  4                  LND=IND+1;
222  4                  CALL FREE$SECTOR;
                 /* LOCATION OF LINKED SECTOR */
223  4                  LINK2(IND).TRK=TRACK$NUMBER;
224  4                  LINK2(IND).SECT=SECTOR$NUMBER;
                 /* NUMBER OF SECTORS TO BE LINKED */
225  4                  NUMSECTOR=NUMSECTOR-1;
226  4                  IF NUMSECTOR=0 THEN GO TO LAST$LINK;
228  4              END;
229  3              CALL FREE$SECTOR;
                 /* LOCATION OF NEXT LINKAGE MAP SECTOR */
230  3              LINK2(1).TRK=TRACK$NUMBER;
231  3              LINK2(1).SECT=SECTOR$NUMBER;
                 /* WRITE CURRENT LINKAGE MAP TO DISK ONE SECTOR AT A TIME */
232  3              CALL LINK$IN$OUT(1,DRIVE,LINK$TRACK,LINK$SECTOR,1);
                 /* REMEMBER CURRENT LINKAGE MAP SECTOR */
233  3              LAST$LINK$SECT=LINK$SECTOR;
234  3              LAST$LINK$TRK=LINK$TRACK;
                 /* CREATE NEW LINK SECTOR */
235  3              LINK$SECTOR=LINK2(1).SECT;
236  3              LINK$TRACK=LINK2(1).TRK;
237  3          END;
238  2          LAST$LINK:
                /* FILL EXTRA LOCATIONS OF LAST LINK SECTOR WITH ZEROES */
              DO INDD=LND TO 63;
239  3              LINK2(INDD).TRK, LINK2(INDD).SECT=0;
240  3          END;
                /* SPECIFY LAST LINKAGE SECTOR */
241  2          LINK2(1).TRK, LINK2(1).SECT=0;
242  2          CALL LINK$IN$OUT(1,DRIVE,LINK$TRACK,LINK$SECTOR,1);/*LAST LINK SECTOR*/
243  2          CALL ABM$IN$OUT(1,DRIVE);          /* WRITE UPDATED ABM */
244  2          RETURN;
```

```
245  2     END MAKE$LINK;

              /*THE FOLLOWING TYPED PROCEDURE RETURNS A VALUE EQUAL TO
              THE NUMBER OF AVAILABLE SECTORS ON DISK DRIVE 'DRIVE'. */
246  1     AVAILABLE$SECTORS:
           PROCEDURE (DRIVE) WORD;
247  2        DECLARE (DRIVE,MASK) BYTE;
248  2        DECLARE COUNT WORD;
249  2        DECLARE (SHIFTS,BYTES) WORD;
250  2        CALL ABM$IN$OUT(0,DRIVE);        /* READ ABM  */
251  2        COUNT=0;
252  2        DO BYTES=0 TO 250;
253  3           MASK=080H;
254  3           DO SHIFTS=1 TO 8;
              /* UPDATE COUNT OF AVAILABLE SECTORS */
255  4                IF (ABM(BYTES) AND MASK)=0 THEN COUNT=COUNT+1;
257  4                MASK=SHR(MASK,1);
258  4           END;
259  3        END;
260  2        RETURN COUNT;
261  2     END AVAILABLE$SECTORS;

              /*THE FOLLOWING PROCEDURE SEARCHES FOR A FILE.  IF THE FILE
              IS FOUND ON THE PROPER DISK, THE SECTOR IN WHICH IT IS FOUND
              AND ITS LOCATION IN DIRECTORYA BUFFER IS RETURNED THROUGH THE
              GLOBALS FILE$SECTORA AND FILEIDENTA.  IF THE FILE IS NOT FOUND,
              THESE GLOBALS ARE SET TO 0FFH.                              */
262  1     FINDFILE:
           PROCEDURE (DRIVE,FILE$NAME$PTR,NAME$EXT$PTR);
263  2        DECLARE DRIVE BYTE;
264  2        DECLARE (FILE$NAME$PTR,NAME$EXT$PTR) POINTER;
265  2        DO FILE$SECTORA=2 TO 26; /* READ DIRECTORY SECTORS (2-26) */
266  3           CALL DIR$IN$OUT(0,DRIVE,FILE$SECTORA,0);
267  3           DO FILE$IDENTA=0 TO 7; /* CHECK EACH DIRECTORY ENTRY */
              /* CURRENT FILE,PROPER FILE NAME AND EXTENSION? ? */
268  4                IF((DIRECTORYA(FILE$IDENTA).FLAG=0) AND
                 ((CMPB(FILE$NAME$PTR,@DIRECTORYA(FILE$IDENTA).FNAME,
                       6)=0FFFFH)) AND
                    ((CMPB(NAME$EXT$PTR,@DIRECTORYA(FILE$IDENTA).NMEXT,
                       3)=0FFFFH)))THEN RETURN;
270  4                IF DIRECTORYA(FILE$IDENTA).FLAG=7FH THEN
271  4                   GO TO EXIT$FILE;
272  4              END;
273  3           END;
274  2        EXIT$FILE:
              FILE$SECTORA,FILE$IDENTA=0FFH; /* VALUES RETURNED IF FILE NOT FOUND */
275  2        RETURN;
276  2     END FINDFILE;

              /*THE FOLLOWING PROCEDURE IS USED TO FIND AN AVIALABLE SPOT
              IN THE DISK'S DIRECTORY.  FILE$IDENTB AND FILE$SECTORB ARE SET
              TO THE LOCATION IN DIRECTORYB BUFFER WHICH IS AVAILABLE AND
              THE PROPER DIRECTORY SECTOR THAT THE BUFFER CORRESPONDS TO.  IF
              NO ROOM IN THE DIRECTORY EXISTS, THE GLOBALS ARE SET TO 0FFH. */
277  1     AVAIL$DIR$ENTRY:
           PROCEDURE (DRIVE);
```

```
278   2          DECLARE DRIVE BYTE;
279   2          DO FILE$SECTORB=2 TO 26;
280   3              CALL DIR$IN$OUT(0,DRIVE,FILE$SECTORB,1);
281   3              DO FILE$IDENTB=0 TO 7;
282   4          DIRECTORYB(FILEIDENTB).FLAG=DIRECTORYB(FILEIDENTB).FLAG+1-1;/*DUMMY*/
                 /* CURRENT FILE? */
283   4                  IF DIRECTORYB(FILE$IDENTB).FLAG<>0 THEN RETURN;
285   4              END;
286   3          END;
287   2          FILE$SECTORB,FILE$IDENTB=0FFH; /* VALUES RETURNED IF DIRECTORY FULL */
288   2          RETURN;
289   2      END AVAIL$DIR$ENTRY;
```

```
                $SUBTITLE('COMMAND ROUTINES')

                   /*THE FOLLOWING PROCEDURE IS USED TO RENAME A FILE ON THE DISK. */
     290   1     RENAME:
                 PROCEDURE (DRIVE,OLDFILENAMEPTR,OLDFILEEXTPTR,NEWFILENAMEPTR,NEWFILEEXTPTR);
     291   2        DECLARE DRIVE BYTE;
     292   2        DECLARE (OLDFILENAMEPTR,OLDFILEEXTPTR) POINTER;
     293   2        DECLARE (NEWFILENAMEPTR,NEWFILEEXTPTR) POINTER;
     294   2        DECLARE (ERROR1,ERROR2) LABEL;
     295   2        CALL FINDFILE(DRIVE,OLDFILENAMEPTR,OLDFILEEXTPTR);
                   /* FILE NOT FOUND? */
     296   2        IF FILE$SECTORA=0FFH THEN GO TO ERROR1;
                   /* WRITE PROTECTED? */
     298   2        IF (DIRECTORYA(FILEIDENTA).ATTR AND 04H) <> 0 THEN GO TO ERROR2;
                   /* CHANGE DIRECTORY */
     300   2        CALL MOVB(NEWFILENAMEPTR,@DIRECTORYA(FILE$IDENTA).FNAME,6);
     301   2        CALL MOVB(NEWFILEEXTPTR,@DIRECTORYA(FILE$IDENTA).NMEXT,3);
                   /* WRITE UPDATED DIRECTORY */
     302   2        CALL DIR$IN$OUT(1,DRIVE,FILESECTORA,0);
     303   2        RETURN;
     304   2        ERROR1:  CALL SIO$NEWLINE;
     305   2        CALL SIO$OUT$STRING(@('FILE NOT FOUND '),15);
     306   2        GO TO NEXT$COMMAND;
     307   2        ERROR2:  CALL SIO$NEWLINE;
     308   2        CALL SIO$OUT$STRING(@('WRITE PROTECTED FILE '),21);
     309   2        GO TO NEXT$COMMAND;
     310   2     END RENAME;

                   /*THE FOLLOWING PROCEDURE IS USED TO CHANGE THE ATTRIBUTES
                   OF A FILE ON A DISK. */
     311   1     ATTRIBUTECHANGE:
                 PROCEDURE (DRIVE,FILE$NAME$PTR,FNAME$EXT$PTR,ORATTRBYTE,ANDATTRBYTE);
     312   2        DECLARE (DRIVE,ORATTRBYTE,ANDATTRBYTE) BYTE;
     313   2        DECLARE (FILE$NAME$PTR,FNAME$EXT$PTR) POINTER;
     314   2        DECLARE ERROR3 LABEL;
     315   2        CALL FINDFILE(DRIVE,FILE$NAME$PTR,FNAME$EXT$PTR);
                   /* FILE NOT FOUND? */
     316   2        IF FILE$SECTORA=0FFH THEN GO TO ERROR3;
     318   2        DIRECTORYA(FILE$IDENTA).ATTR=(DIRECTORYA(FILE$IDENTA).ATTR
                        OR ORATTRBYTE);          /* SET BITS */
     319   2        DIRECTORYA(FILE$IDENTA).ATTR=(DIRECTORYA(FILE$IDENTA).ATTR
                        AND ANDATTRBYTE);        /* RESET BITS */
                   /* WRITE UPDATED DIRECTORY ENTRY */
     320   2        CALL DIR$IN$OUT(1,DRIVE,FILE$SECTORA,0);
     321   2        RETURN;
     322   2        ERROR3:  CALL SIO$NEWLINE;
     323   2        CALL SIO$OUT$STRING(@('FILE NOT FOUND '),15);
     324   2        GO TO NEXT$COMMAND;
     325   2     END ATTRIBUTECHANGE;

                   /* THE FOLLOWING PROCEDURE ERASES A FILE FORM THE DISK. */
     326   1     ERASE:
                 PROCEDURE (DRIVE,FILE$NAME$PTR,FILE$EXT$PTR);
     327   2        DECLARE (DRIVE,LNK$SECT,LNK$TRK) BYTE;
     328   2        DECLARE (FILE$NAME$PTR,FILE$EXT$PTR) POINTER;
     329   2        DECLARE NUMB$SECT WORD;
```

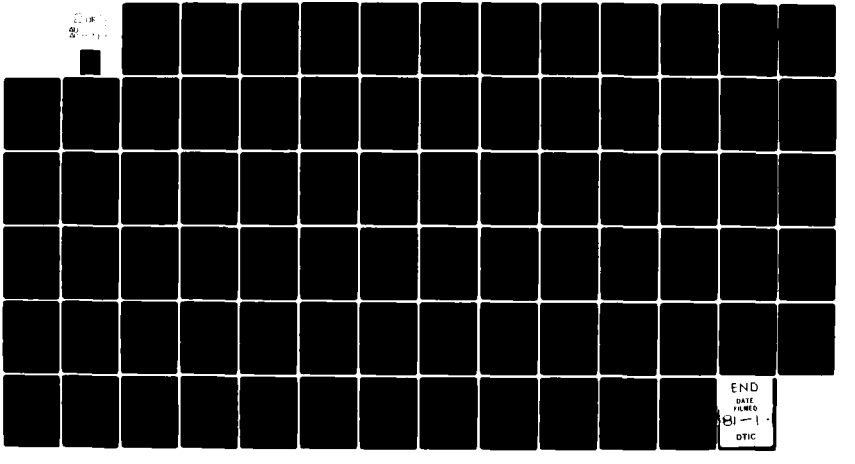
```
330   2         DECLARE I WORD;
331   2         DECLARE (LAST, ERROR4, ERROR5) LABEL;
332   2         CALL FINDFILE(DRIVE, FILE$NAME$PTR, FILE$EXT$PTR);
                /* FILE NOT FOUND? */
333   2         IF FILE$SECTOR#=0FFH THEN GO TO ERROR4;
                /* IS FILE WRITE PROTECTED? */
335   2         IF (DIRECTORY#(FILEIDENT#).ATTR AND 04H) <> 0 THEN GO TO ERROR5;
                /* MAKE FILE NON-CURRENT */
337   2         DIRECTORY#(FILEIDENT#).FLAG=0FFH;
                /* WRITE UPDATED DIRECTORY ENTRY */
338   2         CALL DIR$IN$OUT(1, DRIVE, FILESECTOR#, 0);
339   2         CALL ABM$IN$OUT(0, DRIVE);                    /* READ ABM */
340   2         LNK$SECT=DIRECTORY#(FILE$IDENT#).FST$LNK$SECT;
341   2         LNK$TRK=DIRECTORY#(FILE$IDENT#).FST$LNK$TRK;
342   2         NUMB$SECT=DIRECTORY#(FILE$IDENT#).NUM$SECT;
343   2         DO WHILE TRUE;                                /* UPDATE ABM */
344   3            CALL LINK$IN$OUT(0, DRIVE, LNK$TRK, LNK$SECT, 0);
                      /* MAKE LINK MAP SECTORS AVAILIABLE */
345   3            CALL ABM$ZERO(LNK$TRK, LNK$SECT);
346   3            DO I=2 TO 63;
                        /* MAKE DATA SECTORS AVAILIABLE */
347   4               CALL ABM$ZERO(LINK1(I).TRK, LINK1(I).SECT);
348   4               NUMB$SECT=NUMB$SECT-1;
349   4               IF NUMB$SECT=0 THEN GO TO LAST;
351   4            END;
352   3            LNK$TRK=LINK1(1).TRK;
353   3            LNK$SECT=LINK1(1).SECT;
354   3         END;
355   2         LAST:
                /* WRITE UPDATED ABM TO DISK */
                   CALL ABM$IN$OUT(1, DRIVE);
                   RETURN;
356   2       ERROR4:  CALL SIO$NEWLINE;
357   2         CALL SIO$OUT$STRING(@('FILE NOT FOUND '), 15);
358   2         GO TO NEXT$COMMAND;
359   2       ERROR5:  CALL SIO$NEWLINE;
360   2         CALL SIO$OUT$STRING(@('FILE WRITE PROTECTED '), 21);
361   2         GO TO NEXT$COMMAND;
362   2       END ERASE;

                /*THE FOLLOWING PROCEDURE IS USED TO COPY A FILE FROM ONE
                DISK TO ANOTHER.  */
363   1       KOPY:
                PROCEDURE (FMDRIVE, FMFILENAMEPTR, FMFILEEXTPTR, TODRIVE, TOFILENAMEPTR,
                      TOFILEEXTPTR, INITIALIZE$CALLER);
364   2         DECLARE (FMDRIVE, TODRIVE, INITIALIZE$CALLER) BYTE;
365   2         DECLARE (FMFILENAMEPTR, FMFILEEXTPTR, TOFILENAMEPTR,
                      TOFILEEXTPTR) POINTER;
366   2         DECLARE (FM$LINK$TRK, FM$LINK$SECT, TO$LINK$TRK,
                      TO$LINK$SECT, CHARHOLD) BYTE;
367   2         DECLARE NUMBSECT WORD;
368   2         DECLARE I WORD;
369   2         DECLARE (TRANSFER, ERROR6, ERROR7, ERROR8) LABEL;
                /* JUMP TO TRANSFER IF KOPY WAS CALLED FROM INITIALIZE ROUTINE */
370   2         INITIALIZE$CALLER=INITIALIZE$CALLER+1-1; /*DUMMY*/
371   2         IF INITIALIZE$CALLER<>0 THEN GO TO TRANSFER;
```

```
374   2          CALL FINDFILE(TODRIVE,TOFILENAMEPTR,TOFILEEXTPTR);
                /* DOES FILE ALREADY EXIST? */
375   2          IF FILE$SECTORA<>0FFH THEN
376   2            DO;
377   3              CALL SIO$OUT$STRING(@('FILE ALREADY EXISTS'),19);
378   3              CALL SIO$NEWLINE;
379   3              IF (DIRECTORYA(FILE$IDENTA).ATTR AND 040H) = 040H THEN
380   3               DO;
381   4              CALL SIO$OUT$STRING(@('FILE IS WRITE PROTECTED'),23);
382   4              GO TO ERROR;
383   4               END;
                     ELSE
384   3               DO;
385   4              CALL SIO$OUT$STRING(@('DO YOU WISH NEW FILE TO'),23);
386   4              CALL SIO$NEWLINE;
387   4              CALL SIO$OUT$STRING
                         (@('REPLACE EXISTING FILE (Y OR N)? '),32);
388   4              CALL SIO$GET$CHAR;       /* GET USER RESPONSE */
389   4              CHARHOLD = CHAR;
390   4              CALL SIO$GET$CHAR;
391   4              IF CHAR <> ASCR THEN CALL COMMAND$ERROR;
393   4              IF CHARHOLD = 'N' THEN GO TO NEXT$COMMAND;
395   4              IF CHARHOLD <> 'Y' THEN GO TO ERROR;
                  /* IF USER WANTS TO REPLACE EXISTING FILE THEN ERASE IT. */
397   4              DIRECTORYA(FILE$IDENTA).ATTR = 0;
398   4              CALL DIR$IN$OUT(1,TODRIVE,FILE$SECTORA,0);
399   4              CALL ERASE(TODRIVE,TOFILENAMEPTR,TOFILEEXTPTR);
400   4               END;
401   3            END;
402   2        CALL FINDFILE(FMDRIVE,FMFILENAMEPTR,FMFILEEXTPTR);
                /* FILE NOT FOUND? */
403   2          IF FILE$SECTORA=0FFH THEN GO TO ERROR6;
405   2          CALL AVAIL$DIR$ENTRY(TODRIVE);
                /* IS THERE ENOUGH ROOM IN DIRECTORY? */
406   2          IF FILE$SECTORB=0FFH THEN GO TO ERROR7;
               /* IS THERE ENOUGH ROOM ON THE RECEIVING DISK? */
408   2          TRANSFER: IF DIRECTORYA(FILE$IDENTA).NUM$SECT>AVAILABLE$SECTORS(TODRIVE)
                       THEN GO TO ERROR8;
                /* MAKE LINKAGE MAP ON THE RECEIVING DISK */
410   2          CALL MAKE$LINK(DIRECTORYA(FILE$IDENTA).NUM$SECT,TODRIVE);
                /* SET DIRECTORY ENTRIES */
411   2          DIRECTORYB(FILE$IDENTB).FST$LNK$SECT=FST$LNK$SECT;
412   2          DIRECTORYB(FILE$IDENTB).FST$LNK$TRK=FST$LNK$TRK;
413   2          DIRECTORYB(FILE$IDENTB).FLAG=00;
414   2          DIRECTORYB(FILE$IDENTB).ATTR=DIRECTORYA(FILE$IDENTA).ATTR;
415   2          NUMBSECT,DIRECTORYB(FILE$IDENTB).NUM$SECT=
                       DIRECTORYA(FILE$IDENTA).NUM$SECT;
416   2          DIRECTORYB(FILE$IDENTB).LAST$SECT$BYTE=
                       DIRECTORYA(FILE$IDENTA).LAST$SECT$BYTE;
417   2          CALL MOVB(TOFILENAMEPTR,@DIRECTORYB(FILE$IDENTB).FNAME,6);
418   2          CALL MOVB(TOFILEEXTPTR,@DIRECTORYB(FILE$IDENTB).NMEXT,3);
               /* WRITE UPDATED DIRECTORY TO RECEIVING DISK */
419   2          CALL DIR$IN$OUT(1,TODRIVE,FILE$SECTORB,1);
420   2          FM$LINK$SECT=DIRECTORYA(FILE$IDENTA).FST$LNK$SECT;
               /* DETERMINE FIRST LINKAGE SECTORS */
421   2          FM$LINK$TRK=DIRECTORYA(FILE$IDENTA).FST$LNK$TRK;
```

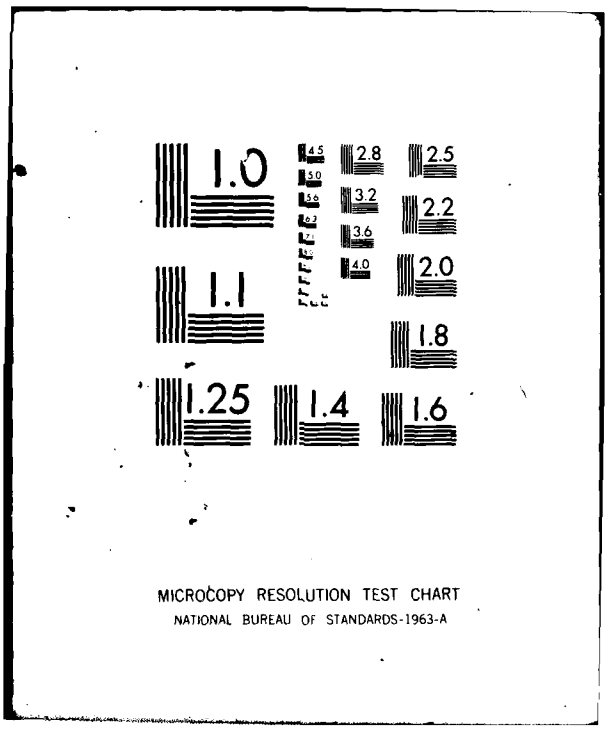
```
422  2          TO$LINK$SECT=DIRECTORYB(FILE$IDENTB).FST$LNK$SECT;
423  2          TO$LINK$TRK=DIRECTORYB(FILE$IDENTB).FST$LNK$TRK;
424  2          DO WHILE TRUE;
                  /* READ LINKAGE SECTORS */
425  3             CALL LINK$IN$OUT(0,FMDRIVE,FM$LINK$TRK,FM$LINK$SECT,0);
426  3             CALL LINK$IN$OUT(0,TODRIVE,TO$LINK$TRK,TO$LINK$SECT,1);
427  3             DO I=2 TO 63;
                      /* READ OLD FILE */
428  4                 CALL DATA$IN$OUT(0,FMDRIVE,LINK1(I).TRK,
                        LINK1(I).SECT,0);
                      /* WRITE NEW FILE */
429  4                 CALL DATA$IN$OUT(1,TODRIVE,LINK2(I).TRK,
                        LINK2(I).SECT,0);
430  4                 NUMBSECT=NUMBSECT-1;
431  4                 IF NUMBSECT=0 THEN RETURN;
433  4               END;
                  /* UPDATE LINKAGE MAP */
434  3               FM$LINK$SECT=LINK1(1).SECT;
435  3               FM$LINK$TRK=LINK1(1).TRK;
436  3               TO$LINK$SECT=LINK2(1).SECT;
437  3               TO$LINK$SECT=LINK2(1).TRK;
438  3             END;
439  2             RETURN;
440  2          ERROR6:  CALL SIO$NEWLINE;
441  2             CALL SIO$OUT$STRING(@('FILE NOT FOUND '),15);
442  2             GO TO NEXT$COMMAND;
443  2          ERROR7:  CALL SIO$NEWLINE;
444  2             CALL SIO$OUT$STRING(@('NO ROOM IN RECEIVING DIRECTORY '),31);
445  2             GO TO NEXT$COMMAND;
446  2          ERROR8:  CALL SIO$NEWLINE;
447  2             CALL SIO$OUT$STRING(@('NOT ENOUGH ROOM ON RECEIVING DISK '),34);
448  2             GO TO NEXT$COMMAND;
449  2       END KOPY;

             /* THE FOLLOWING PROCEDURE WRITES A FILE FROM MEMORY LOCATIONS
             STADD-FNADD TO THE DISK. */
450  1       WRITE:
             PROCEDURE (DRIVE,FILE$NAME$PTR,FILE$EXT$PTR,STADD,FNADD);
451  2          DECLARE (DRIVE,LAST$SECTOR$BYTE,LNKSECT,LNKTRK,CHARHOLD) BYTE;
452  2          DECLARE (FILE$NAME$PTR,FILE$EXT$PTR) POINTER;
453  2          DECLARE (STADD,FNADD,NUM$SECTOR,PTR) WORD;
454  2          DECLARE (LAST,ERROR9,ERROR10) LABEL;
455  2          DECLARE IND WORD;
                /* DETERMINE NUMBER OF SECTORS NEEDED */
456  2          NUM$SECTOR=(FNADD-STADD+1)/128;
457  2          LAST$SECTOR$BYTE=(FNADD-STADD+1) MOD (128);
458  2          IF LAST$SECTOR$BYTE=0 THEN LAST$SECTOR$BYTE=128;
460  2              ELSE NUM$SECTOR=NUM$SECTOR+1;
461  2          IF NUM$SECTOR > AVAILABLE$SECTORS(DRIVE) THEN GO TO ERROR9;
463  2          CALL FINDFILE(DRIVE,FILE$NAME$PTR,FILE$EXT$PTR);
                /* DOES FILE ALREADY EXIST? */
464  2          IF FILE$SECTORA <> 0FFH THEN
465  2             DO;
466  3              CALL SIO$OUT$STRING(@('FILE ALREADY EXISTS'),19);
467  3              CALL SIO$NEWLINE;
468  3              IF (DIRECTORYA(FILEIDENTA).ATTR AND 040H) = 040H THEN
```

```
469  3              DO;
470  4               CALL SIO$OUT$STRING
                         (@('FILE IS WRITE PROTECTED'),23);
471  4               GO TO ERROR;
472  4              END;
                    ELSE
473  3              DO;
474  4               CALL SIO$OUT$STRING
                         (@('DO YOU WISH NEW FILE TO'),23);
475  4               CALL SIO$NEWLINE;
476  4               CALL SIO$OUT$STRING
                         (@('REPLACE EXISTING FILE (Y OR N)?'),31);
477  4               CALL SIO$GET$CHAR;        /* GET USER RESPONSE */
478  4               CHARHOLD=CHAR;
479  4               CALL SIO$GET$CHAR;
480  4               IF CHAR <> ^SCR THEN CALL COMMAND$ERROR;
482  4               IF CHARHOLD = 'N' THEN GO TO NEXT$COMMAND;
484  4               IF CHARHOLD <> 'Y' THEN GO TO ERROR;
                  /* IF USER WANTS TO REPLACE EXISTING FILE THEN ERASE IT */
486  4               DIRECTORYA(FILEIDENTA).ATTR=0;
487  4               CALL DIR$IN$OUT(1,DRIVE,FILESECTORA,0);
488  4               CALL ERASE(DRIVE,FILE$NAME$PTR,FILE$EXT$PTR);
489  4              END;
490  3            END;
491  2          CALL AVAIL$DIR$ENTRY(DRIVE);
                  /* IS THERE ENOUGH ROOM IN THE DIRECTORY? */
492  2          IF FILE$SECTORB=0FFH THEN GO TO ERROR10;
                  /* CREATE DIRECTORY ENTRY */
494  2          DIRECTORYB(FILE$IDENTB).FLAG=0;
495  2          CALL MOVB(FILE$NAME$PTR,@DIRECTORYB(FILE$IDENTB).FNAME,6);
496  2          CALL MOVB(FILE$EXT$PTR,@DIRECTORYB(FILE$IDENTB).FNEXT,3);
497  2          DIRECTORYB(FILE$IDENTB).ATTR=00;
498  2          DIRECTORYB(FILE$IDENTB).NUM$SECT=NUM$SECTOR;
499  2          DIRECTORYB(FILE$IDENTB).LAST$SECT$BYTE=LAST$SECTOR$BYTE;
                  /* MAKE LINKAGE MAP */
500  2          CALL MAKE$LINK(NUM$SECTOR,DRIVE);
501  2          DIRECTORYB(FILE$IDENTB).FST$LNK$SECT=FST$LNK$SECT;
502  2          DIRECTORYB(FILE$IDENTB).FST$LNK$TRK=FST$LNK$TRK;
503  2          CALL DIR$IN$OUT(1,DRIVE,FILE$SECTORB,1);
504  2          PTR=STADD-128;
505  2          LNKSECT=FST$LNK$SECT;
506  2          LNKTRK=FST$LNK$TRK;
                  /* THE DATA IS TRANSFERED IN THIS BLOCK */
507  2          DO WHILE TRUE;
                    /* READ LINKAGE MAP */
508  3           CALL LINK$IN$OUT(0,DRIVE,LNKTRK,LNKSECT,1);
509  3           DO IND=2 TO 63;
510  4              PTR=PTR+128;
511  4              NUM$SECTOR=NUM$SECTOR-1;
512  4              IF NUM$SECTOR=0 THEN GO TO LAST;
                   /* TRANSFER DATA HERE */
514  4              CALL DATA$IN$OUT(1,DRIVE,LINK2(IND).TRK,
                         LINK2(IND).SECT,PTR);
515  4           END;
                  /* GET NEXT LINKAGE MAP SECTOR */
516  3           LNKSECT=LINK2(1).SECT;
```

AD-A091 779 AIR FORCE ACADEMY CO F/G 9/2
USAFA/8086 - A STATE OF THE ART MICROPROCESSOR SYSTEM, VOLUME I--ETC(U)
JUN 80 J J POLLARD
UNCLASSIFIED USAFA-TR-80-16-VOL-2 NL



END
DATE FILMED
81-1-
DTIC


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

```
517  3            LNKTRK=LINK2(1).TRK;
518  3            END;
                /* SET EXCESS BYTES IN LAST DATA SECTOR TO ZERO */
519  2       LAST:  CALL SETB(00,@DDATA,128);
520  2          DPNTR.SEG=0;
521  2          TPNTR.SEG=0;
522  2          DPNTR.OFF=PTR;
523  2          TPNTR.OFF=DDATA$BUF$ADRS;
524  2          CALL MOVB(DATA$POINTER,TARGET$POINTER,LAST$SECTOR$BYTE);
525  2          CALL DATA$IN$OUT(1,DRIVE,LINK2(IND).TRK,LINK2(IND).SECT,0);
526  2          RETURN;
527  2        ERROR9:  CALL SIO$NEWLINE;
528  2          CALL SIO$OUT$STRING(@('NOT ENOUGH ROOM ON DISK '),24);
529  2          GO TO NEXT$COMMAND;
530  2        ERROR10:  CALL SIO$NEWLINE;
531  2          CALL SIO$OUT$STRING(@('NOT ENOUGH ROOM IN THE DIRECTORY '),33);
532  2          GO TO NEXT$COMMAND;
533  2       END WRITE;

             /* THE FOLLOWING PROCEDURE IS USED TO READ A FILE FROM THE DISK
             INTO LOCATIONS STADD-FNADD OF MEMORY. */
534  1       READ:
             PROCEDURE (DRIVE,FILE$NAME$PTR,FILE$EXT$PTR,STADD,FNADD);
535  2          DECLARE (FILE$NAME$PTR,FILE$EXT$PTR) POINTER;
536  2          DECLARE (DRIVE,LINKSECTOR,LINKTRACK,RTRACK,RSECTOR,IND) BYTE;
537  2          DECLARE (STADD,FNADD,SECTORCNT,AVAILSPACE,PTR) WORD;
538  2          DECLARE (LAST,ERROR11,ERROR12) LABEL;
539  2          CALL FINDFILE(DRIVE,FILE$NAME$PTR,FILE$EXT$PTR);
                /* FILE NOT FOUND? */
540  2          IF FILE$SECTOR#=0FFH THEN GO TO ERROR11;
542  2          SECTORCNT=DIRECTORY#(FILE$IDENT#).NUM$SECT;
543  2          AVAILSPACE=FNADD-STADD;
                /* IS THERE ENOUGH ROOM IN RAM? */
544  2          IF (128*(DIRECTORY#(FILE$IDENT#).NUM$SECT)+
                DOUBLE((DIRECTORY#(FILE$IDENT#).LAST$SECT$BYTE))-128) > AVAILSPACE
                   THEN GO TO ERROR12;
546  2          PTR=STADD-128;
547  2          LINKSECTOR=DIRECTORY#(FILE$IDENT#).FST$LNK$SECT;
548  2          LINKTRACK=DIRECTORY#(FILE$IDENT#).FST$LNK$TRK;
549  2          DO WHILE TRUE;
                  /* READ LINKAGE MAP SECTOR */
550  3            CALL LINK$IN$OUT(0,DRIVE,LINKTRACK,LINKSECTOR,0);
551  3            DO IND=2 TO 63;
552  4               RTRACK=LINK1(IND).TRK;
553  4               RSECTOR=LINK1(IND).SECT;
554  4               PTR=PTR+128;
555  4               SECTORCNT=SECTORCNT-1;
                                 /* UPDATE POINTER */
556  4               IF SECTORCNT=0 THEN GO TO LAST;
                    /* TRANSFER THE DATA */
558  4               CALL DATA$IN$OUT(0,DRIVE,RTRACK,RSECTOR,PTR);
559  4            END;
                  /* GET NEXT LINKAGE MAP SECTOR */
560  3            LINKSECTOR=LINK1(1).SECT;
561  3            LINKTRACK=LINK1(1).TRK;
562  3          END;
```

```
563  2          LAST:
                    CALL DATA$IN$OUT(0,DRIVE,RTRACK,RSECTOR,0);
                   /* DONT MOVE THE EXCESS BYTES FROM RAM TO DISK */
564  2             DPNTR.SEG=0;
565  2             TPNTR.SEG=0;
566  2             DPNTR.OFF=DDATA$BUF$ADRS;
567  2             TPNTR.OFF=PTR;
568  2             CALL MOVB(DATA$POINTER,TARGET$POINTER,DIRECTORYA(FILE$IDENTA
                       ).LAST$SECT$BYTE);
569  2           RETURN;
570  2         ERROR11:  CALL SIO$NEWLINE;
571  2           CALL SIO$OUT$STRING(@('FILE NOT FOUND '),15);
572  2           GO TO NEXT$COMMAND;
573  2         ERROR12:  CALL SIO$NEWLINE;
574  2           CALL SIO$OUT$STRING(@('NOT ENOUGH ROOM ALLOCATED IN RAM '),33);
575  2           GO TO NEXT$COMMAND;
576  2       END READ;
```

```
                $SUBTITLE('DRIVER ROUTINES')

                   /*THE FOLLOWING READS IN CHARACTERS UNTIL A NON-BLANK IS FOUND. */
     577   1     SIO$GET$LEAD$BLANKS:
                 PROCEDURE;
     578   2        CALL SIO$GET$CHAR;
     579   2        DO WHILE CHAR = ASBL;
     580   3           CALL SIO$GET$CHAR;
     581   3        END;
     582   2     END SIO$GET$LEAD$BLANKS;

                   /* THE FOLLOWING PROCEDURE READS IN THE DISK DRIVE */
     583   1     SIO$GET$DISK$DRIVE:
                 PROCEDURE BYTE;
     584   2        DECLARE DDRIVE BYTE,DDERR(0R LABEL;
     585   2        CALL SIO$GET$LEAD$BLANKS;
                    /* ENTRY MUST BE IN THE FORM ':F0:' OR ':F1:' OR ELSE AN ERROR
                    MESSAGE IS PRINTED AND THE COMMAND IS ABORTED. */
     586   2        IF CHAR <> ':' THEN GO TO DDERROR;
     588   2        CALL SIO$GET$CHAR;
     589   2        IF CHAR <> 'F' THEN GO TO DDERROR;
     591   2        CALL SIO$GET$CHAR;
     592   2        IF (CHAR <> '0') AND (CHAR <> '1') THEN GO TO DDERROR;
     594   2        DDRIVE=SIO$HEX(CHAR);
     595   2        CALL SIO$GET$CHAR;
     596   2        IF CHAR = ':' THEN RETURN DDRIVE;
     598   2       DDERROR: IF CHAR <> ASCR THEN
     599   2           DO WHILE CHAR <> ASCR;
     600   3            CALL SIO$GET$CHAR;
     601   3           END;
     602   2        CALL SIO$NEWLINE;
     603   2        CALL SIO$OUT$STRING(@('DISK DRIVE FORMAT ERROR'),23);
     604   2        GO TO ERROR;
     605   2     END SIO$GET$DISK$DRIVE;

                   /*THE FOLLOWING PROCEDURE IS USED TO READ IN THE FILENAME
                   AND EXTENSION FROM THE OPERATOR. FILENAME1 AND FILEEXT1 ARE
                   FILLED WITH THE ASCII CHARACTERS OF THE FILENAME. */
     606   1     SIO$GET$FILENAME:
                 PROCEDURE;
     607   2        DECLARE (I,J) INTEGER;
     608   2        DECLARE (FIRSTLETTER,NOEXT,FILEEXT,FLERROR) LABEL;
     609   2        DECLARE TEST1 BYTE;
                    /* THE FILENAME MUST BE 1 TO 6 CHARACTERS LONG.  THE FILENAME
                    EXTENTION CAN BE 1 TO 3 CHARACTERS LONG OR CAN BE OMITTED.
                    THE FILENAME EXTENSION SHOULD BE SEPERATED FROM THE FILENAME
                    WITH A PERIOD ('.').  */
     610   2        CALL USER$IO$GET$CHAR;
     611   2        DO WHILE CHAR=ASBL;
     612   3           CALL USER$IO$GET$CHAR;
     613   3        END;
     614   2        DO I=0 TO 25;     /* INSURE THAT THE FIRST CHARACTER IS A LETTER */
     615   3           IF CHAR=ABC(I) THEN GO TO FIRSTLETTER;
     617   3        END;
     618   2        GO TO FLERROR;
     619   2       FIRSTLETTER: FILENAME1(0)=CHAR;
```

```
620   2         DO I=1 TO 5;
621   3            CALL SIO$GET$CHAR;
622   3            IF CHAR = '.' THEN
623   3               DO;
624   4                  DO J=I TO 5;
                        /* FILL REMAINING LOCATIONS WITH NULLS */
625   5                   FILENAME1(J)=00;
626   5                   END;
627   4                  GO TO FILEEXT;
628   4               END;
629   3            IF (CHAR = ',') OR (CHAR = ASCR) THEN
630   3               DO;
631   4                  DO J=I TO 5;
632   5                     FILENAME1(J)=00;
633   5                  END;
634   4                  GO TO NOEXT;
635   4               END;
636   3            TEST1=0;
                  /* ENSURE THAT ALL CHARACTERS ARE ALPHANUMERIC */
637   3            DO J=0 TO 35;
638   4                IF CHAR = ALPHANUM(J) THEN TEST1=0FFH;
640   4            END;
641   3            TEST1=TEST1+1-1;/*DUMMY STATEMENT*/
642   3            IF TEST1 = 0 THEN GO TO FLERROR;
644   3                  FILENAME1(I)=CHAR;
645   3         END;
646   2         CALL SIO$GET$CHAR;
647   2         IF (CHAR = ',') OR (CHAR = ASCR) THEN
648   2          NOEXT: DO;
649   3               DO I=0 TO 2;
650   4                  FILEEXT1(I) =00;  /* FILL EXTENSION WITH NULLS */
651   4               END;
652   3               RETURN;
653   3            END;
                 /* INSURE THAT A PERIOD SEPERATES THE NAME AND THE EXTENSION */
654   2          IF CHAR <> '.' THEN GO TO FLERROR;
656   2        FILEEXT: DO I=0 TO 2;
657   3            CALL SIO$GET$CHAR;
658   3            IF (CHAR = ',') OR (CHAR = ASCR) THEN
659   3               DO;
660   4                  DO J=I TO 2;
661   5                  FILEEXT1(J) = 00;
662   5                  END;
663   4                  RETURN;
664   4               END;
665   3            TEST1=0;
                  /* INSURE THAT THE FILE EXTENSION IS MADE OF ALPHANUMERICS */
666   3            DO J=0 TO 35;
667   4                IF CHAR = ALPHANUM(J) THEN TEST1=0FFH;
669   4            END;
670   3            TEST1=TEST1+1-1;
671   3            IF TEST1=0 THEN GO TO FLERROR;
672   3               FILEEXT1(I)=CHAR;
674   3         END;
675   2         CALL SIO$GET$CHAR;
                  /* FILENAME ENTRY MUST CONCLUDE WITH A ',' OR A 'RETURN' */
```

```
676  2        IF (CHAR = ',') OR (CHAR = ASCR) THEN RETURN;
678  2      FLERROR: IF CHAR <> ASCR THEN
679  2          DO WHILE CHAR <> ASCR;
680  3            CALL SIO$GET$CHAR;
681  3          END;
682  2        CALL SIO$NEWLINE;
683  2        CALL SIO$OUT$STRING(@('FILE NAME FORMAT ERROR'),22);
684  2        GO TO ERROR;
685  2      END SIO$GET$FILENAME;

           /* THE FOLOWING IS USED TO ENTER THE PARAMETERS FOR THE READ
           AND WRITE COMMANDS. */
686  1      READ$WRITE$PAR$IN:
           PROCEDURE;
687  2        CALL SIO$NEWLINE;
688  2        CALL SIO$OUT$CHAR('*');     /* OUTPUT A PROMPT */
689  2        DRIVE1=SIO$GET$DISK$DRIVE;
690  2        CALL SIO$GET$FILENAME;
691  2        IF CHAR =ASCR THEN
692  2              DO;
           /* THESE ARE THE DEFAULT ADDRESSES IF THEY ARE NOT SPECIFIED */
693  3            STADD = 01000H;
694  3            FNADD = 07FFFH;
695  3            RETURN;
696  3          END;
697  2        CALL SIO$GET$LEAD$BLANKS;
698  2        STADD =SIO$GET$WORD;
699  2        IF CHAR = ASCR THEN
700  2          DO;
            /* THIS IS THE DEFAULT ADDRESS IF ONLY THE START ADD IS ENTERED */
701  3            FNADD=07FFFH;
702  3            RETURN;
703  3          END;
704  2        IF CHAR <> ',' THEN CALL COMMAND$ERROR;
706  2        CALL SIO$GET$LEAD$BLANKS;
707  2        FNADD=SIO$GET$WORD;
708  2        IF CHAR=ASBL THEN CALL SIO$GET$LEAD$BLANKS;
         /* INSURE THAT THE COMMAND IS TERMINATED WITH A 'RETURN' */
710  2        IF CHAR <> ASCR THEN CALL COMMAND$ERROR;
712  2        CALL SIO$NEWLINE;
713  2        RETURN;
714  2        END READ$WRITE$PAR$IN;

           /*THE FOLLOWING INITIATES THE READ COMMAND.
           IT IS CALLED FROM AN EXTERNAL SOURCE. PARAMETERS SHOULD BE ENTERED
           IN THE FORM: ":F0:FNAME.EXT,01000,05000" WHERE 01000 AND 05000 ARE
           THE STARTING AND FINAL ADDRESSES IN RAM. BOTH ADDRESSES OR JUST THE
           FINAL ADDRESS MAY BE OMITTED. */
715  1      READ$DRIVER:
           PROCEDURE PUBLIC;
716  2        CALL SIO$OUT$STRING(@('EAD '),4);
717  2        CALL READ$WRITE$PAR$IN;
718  2        CALL READ(DRIVE1,@FILENAME1,@FILEEXT1,STADD,FNADD);
719  2        CALL SIO$NEWLINE;
720  2        CALL SIO$OUT$STRING(@('READ OPERATION COMPLETED'),24);
721  2        CALL SIO$NEWLINE;
```

```
722  2          END READ$DRIVER;

                /*THE FOLLOWING INITIATES THE WRITE COMMAND.
                IT IS CALLED FROM AN EXTERNAL SOURCE.  PARAMETERS SHOULD BE ENTERED
                AS IN THE EXAMPLE FOR THE 'READ' COMMAND. */
723  1      WRITE$DRIVER:
            PROCEDURE PUBLIC;
724  2          CALL SIO$OUT$STRING(@('RITE '),5);
725  2          CALL READ$WRITE$PAR$IN;
726  2          CALL WRITE(DRIVE1,@FILENAME1,@FILEEXT1,STADD,FNADD);
727  2          CALL SIO$OUT$STRING(@('WRITE OPERATION COMPLETED'),25);
728  2          CALL SIO$NEWLINE;
729  2          END WRITE$DRIVER;

                /*THE FOLLOWING INITIATES THE KOPY COMMAND.
                IT IS CALLED FROM AN EXTERNAL SOURCE.  PARAMETERS SHOULD BE ENTERED
                IN THE FORM: ":F0:FMFILE.EXT,:F1:TOFILE.EXT" */
730  1      KOPY$DRIVER:
            PROCEDURE PUBLIC;
731  2          CALL SIO$OUT$STRING(@('OPY '),4);
732  2          CALL SIO$NEWLINE;
733  2          CALL SIO$OUT$CHAR('*');    /*OUTPUT PROMPT*/
734  2          DRIVE2=SIO$GET$DISK$DRIVE;
735  2          CALL SIO$GET$FILENAME;
736  2          IF CHAR=ASCR THEN CALL COMMAND$ERROR;
738  2          CALL MOVB(@FILENAME1,@FILENAME2,6);
739  2          CALL MOVB(@FILEEXT1,@FILEEXT2,3);
740  2          DRIVE1=SIO$GET$DISK$DRIVE;
741  2          CALL SIO$GET$FILENAME;
               /* ENSURE THE COMMAND TERMINATES WITH A "RETURN." */
742  2          IF CHAR = ',' THEN CALL COMMAND$ERROR;
744  2          CALL KOPY(DRIVE2,@FILENAME2,@FILEEXT2,DRIVE1,@FILENAME1,@FILEEXT1,0);
745  2          END KOPY$DRIVER;

                /*THE FOLLOWING INITIATES THE RENAME COMMAND.
                IT IS CALLED FROM AN EXTERNAL SOURCE.  PARAMETERS SHOULD BE ENTERED
                IN THE FORM ":F0:OLDNME.EXT,NEWNME.EXT"  */
746  1      RENAME$DRIVER:
            PROCEDURE PUBLIC;
747  2          CALL SIO$OUT$STRING(@('AMECHANGE '),10);
749  2          CALL SIO$NEWLINE;
749  2          CALL SIO$OUT$CHAR('*');    /*PROMPT*/
750  2          DRIVE1=SIO$GET$DISK$DRIVE;
751  2          CALL SIO$GET$FILE$NAME;
752  2          IF CHAR <> ',' THEN CALL COMMAND$ERROR;
754  2          CALL MOVB(@FILENAME1,@FILENAME2,6);
755  2          CALL MOVB(@FILEEXT1,@FILEEXT2,3);
756  2          CALL SIO$GET$FILENAME;
               /* ENSURE A "RETURN" TERMINATES THE COMMAND. */
757  2          IF CHAR=' ' THEN CALL COMMAND$ERROR;
759  2          CALL RENAME(DRIVE1,@FILENAME2,@FILEEXT2,@FILENAME1,@FILEEXT1);
760  2              END RENAME$DRIVER;

                /*THE FOLLOWING INITIATES THE ERASE COMMAND.
                IT IS CALLED FROM AN EXTERNAL SOURCE.  PARAMETERS SHOULD BE ENTERED
                IN THE FORM ":F0:FNAME.EXT" */
```

```
761  1      ERASE$DRIVE:
            PROCEDURE PUBLIC;
762  2          CALL SIO$OUT$STRING(@('RASE '),5);
763  2          CALL SIO$NEWLINE;
764  2          CALL SIO$OUT$CHAR('*'); /*PROMPT*/
765  2          DRIVE1=SIO$GET$DISK$DRIVE;
766  2          CALL SIO$GET$FILENAME;
767  2          IF CHAR <> ASCR THEN CALL COMMAND$ERROR;
769  2          CALL ERASE(DRIVE1,@FILENAME1,@FILEEXT1);
770  2          END ERASE$DRIVER;

                /*THE FOLLOWING INITIATES THE ATTRIBUTE$CHANGE COMMAND.
                IT IS CALLED FROM AN EXTERNAL SOURCEL. PARAMETERS SHOULD BE ENTERED
                IN THE FORM ".F0:FNAME.EXT,I,F,NS". THIS WILL MAKE FNAME.EXT AN
                INVISIBLE,FORMAT,NONSYSTEM FILE. THE WRITE-PROTECTION ATTRIBUTE
                WILL NOT BE CHANGED. */
771  1      ATTRIBUTE$CHANGE$DRIVER:
            PROCEDURE PUBLIC;
772  2          DECLARE DONE LABEL;
773  2          DECLARE (ANDATTR,ORATTR) BYTE;
774  2          CALL SIO$OUT$STRING(@('TTRIBUTE CHANGE '),16);
775  2          CALL SIO$NEWLINE;
776  2          CALL SIO$OUT$CHAR('*'); /*PROMPT*/
777  2          DRIVE1=SIO$GET$DISK$DRIVE;
778  2          CALL SIO$GET$FILENAME;
779  2          IF CHAR=ASCR THEN CALL COMMAND$ERROR;
781  2          ORATTR=00H;
782  2          ANDATTR=0FFH;
783  2          DO WHILE TRUE;
784  3              CALL SIO$GET$LEAD$BLANKS;
                   /* SET ATTRIBUTES */
785  3              IF CHAR='I' THEN ORATTR=(ORATTR OR 01H);
787  3              ELSE IF CHAR='W' THEN ORATTR=(ORATTR OR 04H);
789  3              ELSE IF CHAR='F' THEN ORATTR=(ORATTR OR 080H);
791  3              ELSE IF CHAR='S' THEN ORATTR=(ORATTR OR 02H);
793  3              ELSE IF CHAR='N' THEN
794  3                 DO;
                      /* RESET ATTRIBUTES */
795  4                  CALL SIO$GET$CHAR;
796  4                  IF CHAR='I' THEN ANDATTR=(ANDATTR AND 0FEH);
798  4                  ELSE IF CHAR='W' THEN ANDATTR=(ANDATTR AND 0FBH);
800  4                  ELSE IF CHAR='F' THEN ANDATTR=(ANDATTR AND 07FH);
802  4                  ELSE IF CHAR='S' THEN ANDATTR=(ANDATTR AND 0FDH);
804  4                  ELSE CALL COMMAND$ERROR;
805  4                  END;
806  3              ELSE CALL COMMAND$ERROR;
807  3              CALL SIO$GET$CHAR;
                   /* CONTINUE INPUTTING PARAMETERS UNTIL A "RETURN" */
808  3              IF CHAR=ASCR THEN GO TO DONE;
810  3              ELSE IF CHAR <> ',' THEN CALL COMMAND$ERROR;
                END;
813  2          DONE: CALL ATTRIBUTECHANGE(DRIVE1,@FILENAME1,@FILEEXT1,
                   ORATTR,ANDATTR);
814  2          END ATTRIBUTE$CHANGE$DRIVER;

                /*THE FOLLOWING IS USED TO INITIALIZE THE DISK.
```

```
                    IT IS INCLUDED FROM AN EXTERNAL SOURCE */
                $INCLUDE ( F1 IN-C  PLM)
815   1   =     PACK$DATA
            =       PROCEDURE(BYTE0,BYTE1) BYTE;
816   2   =       DECLARE (BYTE0,BYTE1) BYTE;
817   2   =       RETURN SHL(SIO$HEX(BYTE0),4) OR SIO$HEX(BYTE1);
818   2   =       END;
819   1   =     GET$RECORDS:
            =       PROCEDURE(FIRST);
820   2   =       DECLARE (IND,LNGTH,I,ICHK,FIRST,TYPE) BYTE;
821   2   =       DECLARE ADRS WORD;
822   2   =       DECLARE ADRS$PTR POINTER;
823   2   =       DECLARE (AD BASED ADRS$PTR) (2) BYTE;
824   2   =       DECLARE (DATA$SAVED,REC$FOUND) BYTE;
825   2   =       DECLARE (REC BASED TARGET$POINTER) (128) BYTE;
            =
826   2   =       ADRS$PTR=@ADRS;
827   2   =       IND=0;
828   2   =       IF FIRST=1 THEN GO TO START;
830   2   =       ELSE GO TO SET$PARAMETERS;
831   2   =     START:  REC$FOUND=0FFH;
            =       /* FIND RECORD MARK */
832   2   =       DATA$SAVED=0FFH;
833   2   =     GET$REC$MARK:
            =       DO WHILE (DDATA(IND) <> ':') AND (IND < 128);
834   3   =       IND=IND+1;
835   3   =       END;
836   2   =       IF IND=128 THEN RETURN; /* RECORD MARK NOT FOUND */
838   2   =       REC$FOUND=1H;
839   2   =       ICHK=0;
            =       /* CHECK FOR WHOLE RECORD AVAILABLE */
840   2   =       TPNTR.SEG=0;
841   2   =       TPNTR.OFF=DDATA$BUF$ADRS+DOUBLE(IND);
842   2   =       LNGTH=PACK$DATA(REC(1),REC(2));
843   2   =       IF 127< IND+2*LNGTH+10 THEN
844   2   =           DO;
845   3   =           DATA$SAVED=1;
846   3   =           DPNTR.SEG=0;
847   3   =           DPNTR.OFF=DDATA$BUF$ADRS;
848   3   =           TPNTR.SEG=0;
849   3   =           TPNTR.OFF=DIRB$BUF$ADRS;
850   3   =           CALL MOVB(DATA$POINTER,TARGET$POINTER,128);
851   3   =           TPNTR.SEG=0;
852   3   =           TPNTR.OFF=DIRB$BUF$ADRS+DOUBLE(IND);
853   3   =           RETURN;
854   3   =           END;
855   2   =     SET$PARAMETERS:
            =           IF(DATA$SAVED=1) THEN GO TO PROCESS;
857   2   =           IF(REC$FOUND =1) THEN GO TO PROCESS;
859   2   =           GO TO GET$REC$MARK;
860   2   =     PROCESS:LNGTH=PACK$DATA(REC(1),REC(2));
861   2   =       AD(1)=PACK$DATA(REC(3),REC(4));
862   2   =       AD(0)=PACK$DATA(REC(5),REC(6));
863   2   =       TYPE=PACK$DATA(REC(7),REC(8));
864   2   =       ICHK=PACK$DATA(REC(2*LNGTH+9),REC(2*LNGTH+10));
865   2   =       ICHK=ICHK+LNGTH+AD(0)+AD(1)+TYPE;
```

```
866  2  =      DO CASE TYPE;
            =          /* TYPE 0 IS THE DATA RECORD */
867  3  =          DO;
868  4  =          ARG1.OFF=ADRS;
869  4  =          DO I=9 TO 2+LNGTH+7 BY 2;
870  5  =          MEMORY$ARG1=PACK$DATA(REC(I),REC(I+1));
871  5  =          ICHK=ICHK+ MEMORY$ARG1;
872  5  =          ARG1.OFF=ARG1.OFF+1;
873  5  =          END;
874  4  =          END;
            =      /* TYPE 1 RECORD IS AN END OF FILE */
875  3  =          GO TO EXIT$RECORD;
            =      /* TYPE 2 RECORD IS AN EXTENDED ADDRESS RECORD */
876  3  =          DO;
877  4  =          AD(1)=PACK$DATA(REC(9),REC(10));
878  4  =          AD(0)=PACK$DATA(REC(11),REC(12));
879  4  =          ICHK=ICHK+AD(1)+AD(0);
880  4  =          ARG1.SEG=ADRS;
881  4  =          END;
            =      /* TYPE 3 RECORD IS A START ADDRESS RECORD */
882  3  =          DO;
883  4  =          AD(1)=PACK$DATA(REC(9),REC(10));
884  4  =          AD(0)=PACK$DATA(REC(11),REC(12));
885  4  =          ICHK=ICHK+AD(1)+AD(0);
886  4  =          IST.CS=ADRS;
887  4  =          AD(1)=PACK$DATA(REC(13),REC(14));
888  4  =          AD(0)=PACK$DATA(REC(15),REC(16));
889  4  =          ICHK=ICHK+AD(1)+AD(0);
890  4  =          IST.IP=ADRS;
891  4  =          END;
892  3  =      END;
            =
            =          /* CHECK PARITY RESULTS */
893  2  =          ICHK=ICHK+1-1;
894  2  =          IF ICHK<>0 THEN GO TO ERR$PAR;
896  2  =          REC$FOUND=0FFH;
897  2  =          DATA$SAVED=0FFH;
898  2  =          IND=IND+1;
899  2  =          GO TO GET$REC$MARK;
900  2  =  ERR$PAR:
            =          CALL SIO$NEWLINE;
901  2  =          CALL SIO$OUT$STRING(@('PARITY ERROR'),12);
902  2  =          GO TO NEXT$COMMAND;
903  2  =  EXIT$RECORD:    /* CALL START$POINTER */
            =          END GET$RECORDS;
            =
904  1  =  INITIALIZE:
            =          PROCEDURE PUBLIC;
905  2  =          DECLARE (DRIVE,SECTORCNT,LINKSECTOR,LINKTRACK) BYTE;
906  2  =          DECLARE (IND,RTRACK,RSECTOR,LAST,FIRST) BYTE;
            =          /* FINISH LOGO */
907  2  =          CALL SIO$OUT$STRING(@('NPUT HEX FILE ',0DH,0AH,' ' ),16);
908  2  =          DRIVE = SIO$GET$DISK$DRIVE;
909  2  =          CALL SIO$GET$FILENAME;
            =          /* FIND THE FILE */
910  2  =          CALL FINDFILE(DRIVE,@FILENAME1,@FILEEXT1);
```

```
911  2  =          IF FILE$SECTOR#=0FFH THEN GO TO ERROR1;
913  2  =          SECTORCNT=DIRECTORYA(FILE$IDENTA).NUM$SECT;
914  2  =          LINK$SECTOR=DIRECTORYA(FILE$IDENTA).FST$LNK$SECT;
915  2  =          LINK$TRACK=DIRECTORYA(FILE$IDENTA).FST$LNK$TRK;
916  2  =          FIRST=1;
917  2  =          DO WHILE TRUE;
918  3  =             CALL LINK$IN$OUT(0,DRIVE,LINKTRACK,LINKSECTOR,0);
919  3  =             DO IND=2 TO 62;
920  4  =             RTRACK=LINK1(IND).TRK;
921  4  =             RSECTOR=LINK1(IND).SECT;
922  4  =             CALL DATA$IN$OUT(0,DRIVE,RTRACK,RSECTOR,0);
923  4  =             SECTORCNT=SECTORCNT-1;
924  4  =             IF SECTORCNT=0 THEN GO TO LASST;
   =                 /* DATA IN ARRAY DDATA AT 0600H */
926  4  =             CALL GET$RECORDS(FIRST);
927  4  =             FIRST=0FFH;
928  4  =             END;
929  3  =          END;
930  2  =  LASST       DPNTR SEG=0H;
931  2  =          DPNTR OFF=DDATA$BUF$HDRS+
   .  =          DOUBLE(DIRECTORYA(FILE$IDENTA).LAST$SECT$BYTE);
932  2  =          IND=128-DIRECTORYA(FILE$IDENTA).LAST$SECT$BYTE;
933  2  =          CALL SETB(00H,DATA$POINTER,IND);
934  2  =          CALL GET$RECORDS(FIRST);
935  2  =          RETURN;
936  2  =  ERROR1:     CALL SIO$NEWLINE;
937  2  =          CALL SIO$OUT$STRING(@('FILE NOT FOUND'),15);
938  2  =          GO TO NEXT$COMMAND;
939  2  =       END INITIALIZE;
              /*THE FOLLOWING WILL PRINT OUT A DIRECTORY LIST TO
              THE CRT.  IT IS CALLED FROM AN EXTERNAL SOURCE.  FOR INVISIBLE
              FILES AND ATTRIBUTES TO BE SPECIFIED, THE USER MUST ENTER A
              ".A" OR A "0.A" FOR DRIVE 0 OR A "1.A" FOR DRIVE 1. */
940  1     DIRECTORY$LIST:
              PROCEDURE PUBLIC;
941  2        DECLARE NULL BYTE;
942  2        DECLARE TRANSFER LABEL;
943  2        DECLARE (ATTR,DDRIVE,I) BYTE,TOTALSECTORS WORD;
944  2        CALL SIO$OUT$STRING(@(' IRECTORY '),9);
945  2        CALL SIO$GET$LEAD$BLANKS;
              /* DOES FIRST CHARACTER SPECIFY DRIVE? */
946  2        IF (CHAR='0') OR (CHAR='1') THEN
947  2           DO;
948  3           DDRIVE= SIO$HEX$(CHAR);
949  3           CALL SIO$GET$LEAD$BLANKS;
                 /* DOES SECOND CHARACTER SPECIFY NO ATTRIBUTES? */
950  3           IF CHAR=ASCP THEN
951  3             DO;
952  4                ATTR=0;
953  4                GO TO TRANSFER;
954  4             END;
              /* DO SECOND THROUGH FOURTH CHARACTER SPECIFY ATTRIBUTES? */
955  3                 IF CHAR = '.' THEN
956  3             DO;
957  4                CALL SIO$GET$LEAD$BLANKS;
958  4                IF CHAR <> 'A' THEN CALL COMMAND$ERROR;
```

```
960  4              CALL SIO$GET$LEAD$BLANKS;
961  4              IF CHAR <> ASCR THEN CALL COMMAND$ERROR;
963  4              ATTR=0FFH;
964  4              GO TO TRANSFER;
965  4            END;
966  3          END;
                /* DOES FIRST CHARACTER SPECIFY DRIVE 0, NO ATTRIBUTES */
967  2          IF CHAR = ASCR THEN
968  2            DO;
969  3              DDRIVE=0;
970  3              ATTR=0;
971  3              GO TO TRANSFER;
972  3            END;
                /* DO FIRST THROUGH THIRD CHARACTERS SPECIFY DRIVE 0, ATTRIBUTES? */
973  2          IF CHAR = ',' THEN
974  2            DO;
975  3              DDRIVE=0;
976  3              CALL SIO$GET$LEAD$BLANKS;
977  3              IF CHAR <> 'A' THEN CALL COMMAND$ERROR;
979  3              CALL SIO$GET$LEAD$BLANKS;
980  3              IF CHAR <> ASCR THEN CALL COMMAND$ERROR;
982  3              ATTR=0FFH;
983  3              GO TO TRANSFER;
984  3            END;
985  2          CALL COMMAND$ERROR;
986  2        TRANSFER: CALL SIO$NEWLINE;
                 /* OUTPUT HEADER */
987  2          CALL SIO$OUT$STRING(@('FILE    EXT  DATASECTORS'),24);
988  2          IF ATTR= 0FFH THEN CALL SIO$OUT$STRING(@('  ATTR'),6);
990  2          CALL SIO$NEWLINE;
991  2          CALL SIO$NEWLINE;
992  2          NULL=0H;
                /* CHECK ALL DIRECTORY ENTRIES */
993  2          DO FILESECTORB=2 TO 26;
994  3            CALL DIR$IN$OUT(0,DDRIVE,FILESECTORB,1);
995  3            DO FILEIDENTB=0 TO 7;
                /* DETERMINE WHICH FILES SHOULD BE LISTED */
996  4            IF DIRECTORYB(FILE$IDENTB).FLAG=7FH THEN GO TO EXIT$DIR$LIST;
998  4              IF(DIRECTORYB(FILEIDENTB).FLAG=0) AND
                     (((DIRECTORYB(FILEIDENTB).ATTR AND 01H) = 0) OR
                    (ATTR=0FFH)) THEN
999  4                DO;
1000 5                  DO I=0 TO 5;
1001 6                    CALL SIO$OUT$CHAR(DIRECTORYB(FILEIDENTB).FNAME(I));
1002 6                          IF DIRECTORYB(FILEIDENTB).FNAME(I)=NULL THEN
1003 6                       CALL SIO$OUT$CHAR(ASBL);
1004 6                  END;
1005 5                    CALL SIO$BLANKS(2);
1006 5                    DO I=0 TO 2;
1007 6                    CALL SIO$OUT$CHAR(DIRECTORYB(FILEIDENTB).NMEXT(I));
1008 6                    IF DIRECTORYB(FILEIDENTB).NMEXT(I)=NULL THEN
1009 6                       CALL SIO$OUT$CHAR(ASBL);
1010 6                  END;
1011 5                  CALL SIO$BLANKS(6);
1012 5                  CALL SIO$OUT$WORD(DIRECTORYB(FILEIDENTB).NUMSECT);
                           /* DETERMINE WHETHER ATTRIBUTES SHOULD BE LISTED */
```

```
 013   5                IF ATTR<>0FFH THEN
1014   5                 DO;
1015   6                   CALL SIO$BLANK(5);
1016   6                   IF (DIRECTORYB(FILEIDENTB).ATTR AND 02H) <>0 THEN
1017   6                     CALL SIO$OUT$CHAR('S');
1018   6                   IF (DIRECTORYB(FILEIDENTB).ATTR AND 080H) <>0 THEN
1019   6                     CALL SIO$OUT$CHAR('F');
1020   6                   IF (DIRECTORYB(FILEIDENTB).ATTR AND 04H) <>0 THEN
1021   6                     CALL SIO$OUT$CHAR('W');
1022   6                   IF (DIRECTORYB(FILEIDENTB).ATTR AND 01H) <>0 THEN
1023   6                     CALL SIO$OUT$CHAR('I');
1024   6                   END;
1025   5                 CALL SIO$NEWLINE;
1026   5                END;
1027   4              END;
1028   3            END;
1029   2      EXIT$DIR$LIST:
               CALL SIO$OUT$STRING(@('TOTAL SECTORS= '),15);
1030   2           TOTAL$SECTORS= 07D2H - AVAILABLE$SECTORS(DDRIVE);
1031   2           CALL SIO$OUT$WORD(TOTAL$SECTORS);
1032   2           CALL SIO$OUT$STRING(@('/07D2'),5);
1033   2           CALL SIO$NEWLINE;
1034   2           END;
1035   1      END;
```

CROSS-REFERENCE LISTING

DEFN	ADDR	SIZE	NAME, ATTRIBUTES, AND REFERENCES	
63	000CH	26	ABC	BYTE ARRAY(26) DATA 615
54	0680H	256	ABM	BYTE ARRAY(256) AT ABSOLUTE 177 196 204 255
65	0000H	2	ABMBUFADRS	WORD DATA 123 124
189	001EH	2	ABMBYTE	WORD 191 192 196
168	0075H	1	ABMBYTE	BYTE 171 177
120	04D9H	107	ABMINOUT	PROCEDURE STACK=001EH 212 243 250 339 355
169	0012H	2	ABMNUM	WORD 170 171 172
167	062EH	114	ABMZERO	PROCEDURE STACK=0006H 345 347
823	0000H	2	AD	BYTE BASED(ADRSPTR) ARRAY(2) 861 862 865 877 878 879 883 884 885 887 888 889
821	0044H	2	ADRS	WORD 826 868 880 896 890
822	0046H	4	ADRSPTR	POINTER 823 826 861 862 865 877 878 879 883 884 885 887 888 889
64	0026H	36	ALPHANUM	BYTE ARRAY(36) DATA 638 667
773	008FH	1	ANDATTR	BYTE 782 797 799 801 803 813
311	0004H	1	ANDATTRBYTE	BYTE PARAMETER AUTOMATIC 312 319
45	0000H	4	ARG1	STRUCTURE EXTERNAL(17) AT 868 872 880
49			ASBL	LITERALLY 579 611 708 1003 1009

```
 48                    ASCR                    LITERALLY
                                                180  181  391  480  598  599  629  647  658  676  678  679  691  699
                                                710  736  767  779  808  950  961  967  980

943  00A2H     1  ATTR                         BYTE
                                                952  963  970  982  988  998 1013

 50  000AH     1  ATTR                         BYTE MEMBER(DIRECTORYA)
                                                298  318  319  335  379  397  414  468  486

 51  000AH     1  ATTR                         BYTE MEMBER(DIRECTORYB)
                                                414  497  998 1016 1018 1020 1022

311  00AAH   123  ATTRIBUTECHANGE              PROCEDURE STACK=0042H
                                                813

771  1020H   356  ATTRIBUTECHANGEDRIVER        PROCEDURE PUBLIC STACK=0046H

246  00EAH   118  AVAILABLESECTORS             PROCEDURE WORD STACK=0024H
                                                408  461 1030

277  0A4DH   141  AVAILDIRENTRY                PROCEDURE STACK=0028H
                                                405  491

537  003CH     2  AVAILSPACE                   WORD
                                                543  544

 10  0000H     1  B                            BYTE PARAMETER
                                                 11

815  0006H     1  BYTE0                        BYTE PARAMETER AUTOMATIC
                                                816  817

815  0004H     1  BYTE1                        BYTE PARAMETER AUTOMATIC
                                                816  817

188  0018H     2  BYTECOUNT                    WORD
                                                192  201  202  203  204

249  002AH     2  BYTES                        WORD
                                                252  255

  2  0000H     1  C                            BYTE PARAMETER
                                                  3

  7  0000H     1  C                            BYTE PARAMETER
                                                  8

 24  0000H     1  C                            BYTE PARAMETER
                                                 25

 47  0000H     1  CHAR                         BYTE EXTERNAL(20)
                                                180  181  389  391  478  480  579  586  589  592  594  596  598  599
                                                611  615  619  622  629  638  644  647  654  658  667  673  676  678
                                                679  691  699  704  708  710  736  742  752  757  767  779  785  787
                                                789  791  793  796  798  800  802  808  810  946  948  950  955  958
```

```
                                        961  967  973  977  980

  367  0083H     1  CHARHOLD             BYTE
                                        389  393  395

  451  0087H     1  CHARHOLD             BYTE
                                        478  482  484

                  CMPB                  BUILTIN
                                        268

  179  06A0H    46  COMMANDERROR         PROCEDURE STACK=0008H
                                        392  481  705  711  737  743  753  758  768  780  804  806  811  959
                                        962  978  981  985

  248  0026H     2  COUNT                WORD
                                        251  256  260

   68  0002H     2  CS                   WORD MEMBER(IST)
                                        886

  149  0004H     2  DATAADRESS           WORD PARAMETER AUTOMATIC
                                        151  153  154  160  161

  149  05BAH   116  DATAINOUT            PROCEDURE STACK=0024H
                                        428  429  514  525  558  563  922

   66  0006H     4  DATAPOINTER          POINTER
                                         66  524  568  850  933

  824  0096H     1  DATASAVED            BYTE
                                        832  845  855  897

   55  0600H   128  DDATA                BYTE ARRAY(128) AT ABSOLUTE
                                        519  833

   65  000AH     2  DDATABUFADRS         WORD DATA
                                        156  157  523  566  841  847  931

  584  1725H        DDERROR              LABEL
                                        587  590  593  598

  584  008DH     1  DDRIVE               BYTE
                                        594  597

  943  00A3H     1  DDRIVE               BYTE
                                        948  969  975  994 1030

   31  0000H     2  DEFAULTBASE          WORD PARAMETER
                                         32

    2  0000H        DELAY                PROCEDURE EXTERNAL(0) STACK=0000H

    5  0000H        DELAYLONG            PROCEDURE EXTERNAL(1) STACK=0000H

   60  0000H     1  DERR                 BYTE EXTERNAL(23)
```

87

65	0000H	2	DIRBUFADRS	WORD DATA 109 110
65	0002H	2	DIRBUFADRS	WORD DATA 113 114 849 852
50	0500H	128	DIRECTORYA	STRUCTURE ARRAY(8) AT ABSOLUTE 268 270 298 300 301 318 319 335 337 340 341 342 379 397 408 410 414 415 416 420 421 468 486 542 544 547 548 568 913 914 915 931 932
51	0580H	128	DIRECTORYB	STRUCTURE ARRAY(8) AT ABSOLUTE 282 283 411 412 413 414 415 416 417 418 422 423 494 495 496 497 498 499 501 502 996 998 1001 1002 1007 1008 1012 1016 1018 1020 1022
940	2374H	942	DIRECTORYLIST	PROCEDURE PUBLIC STACK=0028H
103	0465H	116	DIRINOUT	PROCEDURE STACK=0022H 266 280 302 320 338 398 419 487 503 994
69	0000H		DISK	PROCEDURE EXTERNAL(24) STACK=0000H 117 126 129 146 164
772	1E66H		DONE	LABEL 809 813
1	035AH		DOS	PROCEDURE STACK=0000H
			DOUBLE	BUILTIN 170 544 841 852 931
66	0006H	4	DPNTR	STRUCTURE AT 520 522 564 566 846 847 930 931
94	0004H	1	DRIVE	BYTE PARAMETER AUTOMATIC 95 100
120	0004H	1	DRIVE	BYTE PARAMETER AUTOMATIC 121 122
246	0004H	1	DRIVE	BYTE PARAMETER AUTOMATIC 247 250
206	0004H	1	DRIVE	BYTE PARAMETER AUTOMATIC 208 212 232 242 243
277	0004H	1	DRIVE	BYTE PARAMETER AUTOMATIC 278 280
326	000CH	1	DRIVE	BYTE PARAMETER AUTOMATIC 327 332 338 339 344 355
149	000AH	1	DRIVE	BYTE PARAMETER AUTOMATIC

```
                                        150  152

    132  0084H     1  DRIVE             BYTE PARAMETER AUTOMATIC
                                        133  134

    534  0010H     1  DRIVE             BYTE PARAMETER AUTOMATIC
                                        536  539  550  558  563

    450  0010H     1  DRIVE             BYTE PARAMETER AUTOMATIC
                                        451  461  463  487  488  491  500  503  508  514  525

    103  0008H     1  DRIVE             BYTE PARAMETER AUTOMATIC
                                        104  105

    905  0098H     1  DRIVE             BYTE
                                        908  910  918  922

    311  0010H     1  DRIVE             BYTE PARAMETER AUTOMATIC
                                        312  315  320

    262  000CH     1  DRIVE             BYTE PARAMETER AUTOMATIC
                                        263  266

    290  0014H     1  DRIVE             BYTE PARAMETER AUTOMATIC
                                        291  295  302

     57  0054H     1  DRIVE1            BYTE
                                        689  718  726  740  744  750  759  765  769  777  813

     57  0055H     1  DRIVE2            BYTE
                                        734  744

    326  0C25H   341  ERASE             PROCEDURE STACK=003EH
                                        399  488  769

    761  1CCAH    86  ERASEDRIVER       PROCEDURE PUBLIC STACK=0042H

     46  0000H        ERROR             LABEL EXTERNAL(18)
                                        184  382  396  471  485  604  684

    936  2358H        ERROR1            LABEL
                                        912

    294  0B74H        ERROR1            LABEL
                                        297  304

    454  14ABH        ERROR10           LABEL
                                        493  530

    538  1648H        ERROR11           LABEL
                                        541  570

    538  1664H        ERROR12           LABEL
                                        545  573

    294  088DH        ERROR2            LABEL
```

299 307

314 0C88H ERROR3 LABEL
317 322

331 0D44H ERROR4 LABEL
334 357

331 0D5DH ERROR5 LABEL
336 360

370 1106H ERROR6 LABEL
404 440

370 111FH ERROR7 LABEL
407 443

370 1139H ERROR8 LABEL
409 446

454 1492H ERROR9 LABEL
462 527

85 03E2H 79 ERRORCHECKER PROCEDURE STACK=000EH
118 127 130 147 165

900 21F9H ERRPAR LABEL
895

1029 26E3H EXITDIRLIST LABEL
997

274 0A3FH EXITFILE LABEL
271

903 2212H EXITRECORD LABEL
875

608 1920H FILEEXT LABEL
627 656

58 0062H 3 FILEEXT1 BYTE ARRAY(3)
650 661 673 718 726 739 744 755 759 769 813 910

58 0065H 3 FILEEXT2 BYTE ARRAY(3)
739 744 755 759

534 0008H 4 FILEEXTPTR POINTER PARAMETER AUTOMATIC
535 539

450 0008H 4 FILEEXTPTR POINTER PARAMETER AUTOMATIC
452 463 488 496

326 0004H 4 FILEEXTPTR POINTER PARAMETER AUTOMATIC
328 332

```
 57  0040H    1  FILEIDENTA            BYTE
                                        267  268  270  274  298  300  301  318  319  335  337  340  341  342
                                        379  397  408  410  414  415  416  420  421  468  486  542  544  547
                                        548  568  913  914  915  931  932

 57  004FH    1  FILEIDENTB            BYTE
                                        281  282  283  287  411  412  413  414  415  416  417  418  422  423
                                        494  495  496  497  498  499  501  502  995  996  998 1001 1002 1007
                                       1008 1012 1016 1018 1020 1022

 58  0056H    6  FILENAME1             BYTE ARRAY(6)
                                        619  625  632  644  718  726  738  744  754  759  769  813  910

 58  005CH    6  FILENAME2             BYTE ARRAY(6)
                                        738  744  754  759

262  0008H    4  FILENAMEPTR           POINTER PARAMETER AUTOMATIC
                                        264  268

534  000CH    4  FILENAMEPTR           POINTER PARAMETER AUTOMATIC
                                        535  539

450  000CH    4  FILENAMEPTR           POINTER PARAMETER AUTOMATIC
                                        452  463  488  495

326  0008H    4  FILENAMEPTR           POINTER PARAMETER AUTOMATIC
                                        328  332

311  000CH    4  FILENAMEPTR           POINTER PARAMETER AUTOMATIC
                                        313  315

 57  004CH    1  FILESECTORA           BYTE
                                        265  266  274  296  302  316  320  333  338  375  398  403  464  487
                                        540  911

 57  004EH    1  FILESECTORB           BYTE
                                        279  280  287  406  419  492  503  993  994

 71  035AH  136  FILLIOPB              PROCEDURE STACK=0016H
                                        116  125  128  145  163

262  0960H  237  FINDFILE              PROCEDURE STACK=0030H
                                        295  315  332  374  402  463  539  910

819  0004H    1  FIRST                 BYTE PARAMETER AUTOMATIC
                                        820  828

906  00A0H    1  FIRST                 BYTE
                                        916  926  927  934

608  17B7H       FIRSTLETTER           LABEL
                                        616  619

 51  0000H    1  FLAG                  BYTE MEMBER(DIRECTORYB)
                                        282  283  413  494  996  998
```

```
  50  0000H    1  FLAG                BYTE MEMBER(DIRECTORYA)
                                       268  270  337

 608  1A14H       FLERROR             LABEL
                                       618  643  655  672  678

 364  0010H    1  FMDRIVE             BYTE PARAMETER AUTOMATIC
                                       365  402  425  428

 364  0010H    4  FMFILEEXTPTR        POINTER PARAMETER AUTOMATIC
                                       366  402

 364  0014H    4  FMFILENAMEPTR       POINTER PARAMETER AUTOMATIC
                                       366  402

 367  0000H    1  FMLINKSECT          BYTE
                                       420  425  434

 367  007FH    1  FMLINKTRK           BYTE
                                       421  425  435

 534  0004H    2  FNADD               WORD PARAMETER AUTOMATIC
                                       537  543

 450  0004H    2  FNADD               WORD PARAMETER AUTOMATIC
                                       453  456  457

  57  0002H    2  FNADD               WORD
                                       694  701  707  718  726

  51  0001H    6  FNAME               BYTE ARRAY(6) MEMBER(DIRECTORYB)
                                       417  495 1001 1002

  50  0001H    6  FNAME               BYTE ARRAY(6) MEMBER(DIRECTORYA)
                                       268  300

 311  0000H    4  FNAMEEXTPTR         POINTER PARAMETER AUTOMATIC
                                       313  315

 190  073AH       FOUND               LABEL
                                       197  201

 186  06CEH  169  FREESECTOR          PROCEDURE STACK=0004H
                                       214  222  229

  57  0050H    1  FSTLNKSECT          BYTE
                                       215  411  501  505

  50  000EH    1  FSTLNKSECT          BYTE MEMBER(DIRECTORYA)
                                       340  420  547  914

  51  000EH    1  FSTLNKSECT          BYTE MEMBER(DIRECTORYB)
                                       411  422  501

  51  000FH    1  FSTLNKTRK           BYTE MEMBER(DIRECTORYB)
                                       412  423  502
```

```
 50  000FH    1  FSTLNKTRK          BYTE MEMBER(DIRECTORYA)
                                     341  421  548  915

 57  0051H    1  FSTLNKTRK          BYTE
                                     216  412  502  506

 60  0000H    1  FTERR              BYTE EXTERNAL(22)
                                      86   87

833  1ED3H       GETRECMARK         LABEL
                                     859  899

819  1EA3H  883  GETRECORDS         PROCEDURE STACK=0016H
                                     926  934

 21  0000H    1  H                  BYTE PARAMETER
                                      22

                 HIGH               BUILTIN
                                     110  114  124  139  143  157  161

607  0040H    2  I                  INTEGER
                                     614  615  620  624  631  644  649  659  656  660  673

169  0016H    2  I                  WORD
                                     174

820  0093H    1  I                  BYTE
                                     869  870

943  00A4H    1  I                  BYTE
                                    1000 1001 1002 1006 1007 1008

369  0032H    2  I                  WORD
                                     427  428  429

330  002EH    2  I                  WORD
                                     346  347

820  0094H    1  ICHK               BYTE
                                     839  864  865  871  879  885  889  893  894

132  0004H    1  IDENT              BYTE PARAMETER AUTOMATIC
                                     133  135  136

103  0004H    1  IDENT              BYTE PARAMETER AUTOMATIC
                                     104  106  107

906  009CH    1  IND                BYTE
                                     919  920  921  932  933

536  008CH    1  IND                BYTE
                                     551  552  553

820  0091H    1  IND                BYTE
                                     827  833  834  836  841  843  852  898
```

```
210  0020H    2  IND                  WORD
                                       220  221  223  224

455  0038H    2  IND                  WORD
                                       509  514  525

210  0022H    2  INDD                 WORD
                                       238  239

904  2216H  350  INITIALIZE           PROCEDURE PUBLIC STACK=0034H

364  0004H    1  INITIALIZECALLER     BYTE PARAMETER AUTOMATIC
                                       365  371  372

149  000CH    1  INOUT                BYTE PARAMETER AUTOMATIC
                                       150  152

132  000CH    1  INOUT                BYTE PARAMETER AUTOMATIC
                                       133  134

120  0006H    1  INOUT                BYTE PARAMETER AUTOMATIC
                                       121  122

103  000AH    1  INOUT                BYTE PARAMETER AUTOMATIC
                                       104  105

 94  0006H    1  INOUT                BYTE PARAMETER AUTOMATIC
                                        95   96   97

 61  0700H   10  IOBP                 BYTE ARRAY(10) AT ABSOLUTE
                                        73   74   75   76   77   78   79   80   81   82

 59  0000H    2  IOPB                 WORD EXTERNAL(21)
                                        83

 94  0431H   52  IOPBONE              PROCEDURE BYTE STACK=0006H
                                       105  122  134  152

 68  0000H    2  IP                   WORD MEMBER(IST)
                                       890

 68  000EH    4  IST                  STRUCTURE AT
                                       886  890

607  0042H    2  J                    INTEGER
                                       624  625  631  632  637  638  660  661  666  667

364  0D79H  987  KOPY                 PROCEDURE STACK=0058H
                                       744

730  1B90H  164  KOPYDRIVER           PROCEDURE PUBLIC STACK=005CH

930  2308H       LASST                LABEL
                                       925

906  009FH    1  LAST                 BYTE
```

```
538  15F0H        LAST                    LABEL
                                          557  563

454  142BH        LAST                    LABEL
                                          513  519

331  0D37H        LAST                    LABEL
                                          350  355

 62  0004H    2   LASTABMBYTE             WORD
                                          203

211  007AH        LASTLINK                LABEL
                                          227  238

208  0078H    1   LASTLINKSECT            BYTE
                                          213  219  233

208  0079H    1   LASTLINKTRK             BYTE
                                          213  218  234

 51  0008H    1   LASTSECTBYTE            BYTE MEMBER(DIRECTORYB)
                                          416  499

 50  0008H    1   LASTSECTBYTE            BYTE MEMBER(DIRECTORYA)
                                          416  544  568  931  932

451  0004H    1   LASTSECTORBYTE          BYTE
                                          457  458  459  499  524

 52  0400H  128   LINK1                   STRUCTURE ARRAY(64) AT ABSOLUTE
                                          347  352  353  428  434  435  552  553  560  561  920  921

 65  0004H    2   LINK1BUFADRS            WORD DATA
                                          138  139

 53  0480H  128   LINK2                   STRUCTURE ARRAY(64) AT ABSOLUTE
                                          218  219  223  224  230  231  235  236  239  241  429  436  437  514
                                          516  517  525

 65  0006H    2   LINK2BUFADRS            WORD DATA
                                          142  143

132  0544H  118   LINKINOUT               PROCEDURE STACK=0024H
                                          232  242  344  425  426  508  550  918

905  009AH    1   LINKSECTOR              BYTE
                                          914  918

536  0008H    1   LINKSECTOR              BYTE
                                          547  550  560

208  007AH    1   LINKSECTOR              BYTE
                                          215  232  233  235  242

209  007BH    1   LINKTRACK               BYTE
```

```
                                        216  232  234  236  242

      905  0098H     1  LINKTRACK        BYTE
                                        915  918

      536  0089H     1  LINKTRACK        BYTE
                                        548  550  561

      210  0024H     2  LND              WORD
                                        221  238

      820  0092H     1  LNGTH            BYTE
                                        842  843  860  864  865  869

      451  0085H     1  LNKSECT          BYTE
                                        505  508  516

      327  007DH     1  LNKSECT          BYTE
                                        340  344  345  353

      451  0086H     1  LNKTRK           BYTE
                                        506  508  517

      327  007EH     1  LNKTRK           BYTE
                                        341  344  345  352

                     LOW                BUILTIN
                                        109  113  123  138  142  156  160  201  202

      206  0777H   371  MAKELINK         PROCEDURE STACK=002CH
                                        410  500

      247  007CH     1  MASK             BYTE
                                        253  255  257

      187  0077H     1  MASK             BYTE
                                        193  196  198  204

      168  0076H     1  MASK             BYTE
                                        173  175  177

       45  0000H     1  MEMORYARG1       BYTE BASED(MEMORYARG1PTR)
                                        870  871

       45  0000H     4  MEMORYARG1PTR    POINTER EXTERNAL(17)
                                         45  870  871

                     MOVB               BUILTIN
                                        300  301  417  418  495  496  524  568  738  739  754  755  850

       34  0000H     1  N                BYTE PARAMETER
                                         35

       16  0000H     1  N                BYTE PARAMETER
                                         17
```

```
262  0004H  4  NAMEEXTPTR          POINTER PARAMETER AUTOMATIC
                                   264  268

290  0004H  4  NEWFILEEXTPTR       POINTER PARAMETER AUTOMATIC
                                   293  301

290  0008H  4  NEWFILENAMEPTR      POINTER PARAMETER AUTOMATIC
                                   293  300

 46  0000H     NEXTCOMMAND         LABEL EXTERNAL(19)
                                    90  306  309  324  359  362  394  442  445  448  483  529  532  572
                                   575  902  938

 51  0007H  3  NMEXT              BYTE ARRAY(3) MEMBER(DIRECTORYB)
                                   418  496 1007 1008

 50  0007H  3  NMEXT              BYTE ARRAY(3) MEMBER(DIRECTORYA)
                                   268  301

600  18EBH     NOEXT               LABEL
                                   634  648

941  00A1H  1  NULL                BYTE
                                   992 1002 1008

368  0030H  2  NUMBSECT           WORD
                                   415  430  431

329  002CH  2  NUMBSECT           WORD
                                   342  348  349

 51  000CH  2  NUMSECT            WORD MEMBER(DIRECTORYB)
                                   415  493 1012

 50  000CH  2  NUMSECT            WORD MEMBER(DIRECTORYA)
                                   342  408  410  415  542  544  913

206  0006H  2  NUMSECTOR          WORD PARAMETER AUTOMATIC
                                   207  225  226

453  0034H  2  NUMSECTOR          WORD
                                   456  460  461  498  500  511  512

 67  0000H  2  OFF                WORD MEMBER(TPNTR)
                                   523  567  841  849  852

 66  0000H  2  OFF                WORD MEMBER(DPNTR)
                                   522  566  847  931

 45  0000H  2  OFF                WORD MEMBER(ARG1)
                                   868  872

290  000CH  4  OLDFILEEXTPTR       POINTER PARAMETER AUTOMATIC
                                   292  295

290  0010H  4  OLDFILENAMEPTR      POINTER PARAMETER AUTOMATIC
```

```
                                    292  295

     773  0090H    1  ORATTR         BYTE
                                     781  786  788  790  792  813

     311  0006H    1  ORATTRBYTE     BYTE PARAMETER AUTOMATIC
                                     312  318

     150  0072H    1  P1             BYTE
                                     152  163

     133  006FH    1  P1             BYTE
                                     134  145

     121  006CH    1  P1             BYTE
                                     122  125  128

     104  0069H    1  P1             BYTE
                                     105  116

     150  0073H    1  P5             BYTE
                                     156  160  163

     133  0070H    1  P5             BYTE
                                     138  142  145

     121  006DH    1  P5             BYTE
                                     123  125

     104  006AH    1  P5             BYTE
                                     109  113  116

     150  0074H    1  P6             BYTE
                                     157  161  163

     133  0071H    1  P6             BYTE
                                     139  143  145

     121  006EH    1  P6             BYTE
                                     124  125

     104  006BH    1  P6             BYTE
                                     110  114  116

     815  1E84H   31  PACKDATA       PROCEDURE BYTE STACK=0010H
                                     842  860  861  862  863  864  870  877  878  883  884  887  888

     860  1FD1H       PROCESS        LABEL
                                     856  858

     537  003EH    2  PTR            WORD
                                     546  554  558  567

     453  0036H    2  PTR            WORD
                                     504  510  514  522
```

```
534  14C8H  441  READ                PROCEDURE STACK=0042H
                                      718

715  1AF1H   82  READDRIVER           PROCEDURE PUBLIC STACK=0046H

686  1A53H  158  READWRITEPARIN       PROCEDURE STACK=0012H
                                      717  725

825  0000H  128  REC                 BYTE BASED(TARGETPOINTER) ARRAY(128)
                                      842  860  861  862  863  864  870  877  878  883  884  887  888

824  0097H    1  RECFOUND            BYTE
                                      831  838  857  896

290  0ADAH  208  RENAME              PROCEDURE STACK=0046H
                                      759

746  1C34H  150  RENAMEDRIVER        PROCEDURE PUBLIC STACK=004AH

                 ROR                 BUILTIN
                                      175

906  009EH    1  RSECTOR             BYTE
                                      921  922

536  0088H    1  RSECTOR             BYTE
                                      553  558  563

906  009DH    1  RTRACK              BYTE
                                      920  922

536  0087H    1  RTRACK              BYTE
                                      552  558  563

 53  0000H    1  SECT                BYTE MEMBER(LINK2)
                                      219  224  231  235  239  241  429  436  514  516  525

 52  0000H    1  SECT                BYTE MEMBER(LINK1)
                                      347  353  428  434  553  560  921

167  0004H    1  SECTOR              BYTE PARAMETER AUTOMATIC
                                      168  170

149  0006H    1  SECTOR              BYTE PARAMETER AUTOMATIC
                                      150  163

132  0006H    1  SECTOR              BYTE PARAMETER AUTOMATIC
                                      133  145

103  0006H    1  SECTOR              BYTE PARAMETER AUTOMATIC
                                      104  116

905  0099H    1  SECTORCNT           BYTE
                                      913  923  924

537  0083H    2  SECTORCNT           WORD
```

```
                                        542  555  556

     57  0052H    1  SECTORNUMBER        BYTE
                                         201  215  224  231

     45  0002H    2  SEG                 WORD MEMBER(ARG1)
                                         880

     67  0002H    2  SEG                 WORD MEMBER(TPNTR)
                                         521  565  840  848  851

     66  0002H    2  SEG                 WORD MEMBER(DPNTR)
                                         520  564  846  930

                    SETB                 BUILTIN
                                         519  933

    855  1FB4H      SETPARAMETERS        LABEL
                                         830

    188  001AH    2  SETCOUNT            WORD
                                         195  201  202

    249  0028H    2  SHIFTS              WORD
                                         254

    189  001CH    2  SHIFTS              WORD
                                         194  195

    169  0014H    2  SHIFTS              WORD
                                         172  174

                    SHL                  BUILTIN
                                         100  817

                    SHR                  BUILTIN
                                         198  257

     34  0000H      SIOBLANKS            PROCEDURE EXTERNAL(12) STACK=0000H
                                         1005 1011 1015

     37  0000H      SIOCRLF              PROCEDURE EXTERNAL(13) STACK=0000H

     31  0000H      SIOGETADDR           PROCEDURE EXTERNAL(11) STACK=0000H

     27  0000H      SIOGETBYTE           PROCEDURE BYTE EXTERNAL(9) STACK=0000H

     19  0000H      SIOGETCHAR           PROCEDURE EXTERNAL(6) STACK=0000H
                                         182  388  390  477  479  578  580  588  591  595  600  621  646  657
                                         675  680  795  807

    583  16A1H  195  SIOGETDISKDRIVE     PROCEDURE BYTE STACK=000EH
                                         689  734  740  750  765  777  988

    606  1764H  751  SIOGETFILENAME      PROCEDURE STACK=000EH
                                         690  735  741  751  756  766  778  989
```

```
  577  1681H   32  SIOGETLEADBLANKS      PROCEDURE STACK=0000H
                                          585  697  706  709  784  945  949  957  960  976  979

   29  0000H       SIOGETWORD           PROCEDURE WORD EXTERNAL(10) STACK=0000H
                                          698  707

   24  0000H       SIOHEX               PROCEDURE BYTE EXTERNAL(8) STACK=0000H
                                          594  817  948

   39  0000H       SIONEWLINE           PROCEDURE EXTERNAL(14) STACK=0000H
                                          304  307  322  357  360  378  386  440  443  446  467  475  527  530
                                          570  573  602  682  687  712  719  721  728  732  748  763  775  900
                                          936  986  990  991 1025 1033

   10  0000H       SIOOUTBYTE           PROCEDURE EXTERNAL(3) STACK=0000H

    7  0000H       SIOOUTCHAR           PROCEDURE EXTERNAL(2) STACK=0000H
                                          688  733  749  764  776 1001 1003 1007 1009 1017 1019 1021 1023

   16  0000H       SIOOUTSTRING         PROCEDURE EXTERNAL(5) STACK=0000H
                                           89  305  308  323  358  361  377  381  385  387  441  444  447  466
                                          470  474  476  528  531  571  574  603  683  716  720  724  727  731
                                          747  762  774  901  907  937  944  987  989 1029 1032

   13  0000H       SIOOUTWORD           PROCEDURE EXTERNAL(4) STACK=0000H
                                         1012 1031

   21  0000H       SIOVALIDHEX          PROCEDURE EXTERNAL(7) STACK=0000H

  534  0006H   2   STADD                WORD PARAMETER AUTOMATIC
                                          537  543  546

  450  0006H   2   STADD                WORD PARAMETER AUTOMATIC
                                          453  456  457  504

   57  0000H   2   STADD                WORD
                                          693  698  718  726

  831  1EC9H       START                LABEL
                                          829

   68  000EH   4   STARTPOINTER          POINTER
                                           68

   16  0000H   4   STRPTR               POINTER PARAMETER
                                           17

   41  0000H       SUPPRESSBLANKS       PROCEDURE EXTERNAL(15) STACK=0000H

   67  000AH   4   TARGETPOINTER         POINTER
                                           67  524  568  825  842  850  860  861  862  863  864  870  877  878
                                          883  884  887  888

  609  000EH   1   TEST1                BYTE
                                          636  639  641  642  665  668  670  671
```

```
364  000EH    1  TODRIVE              BYTE PARAMETER AUTOMATIC
                                       365  374  398  399  405  408  410  419  426  429

364  0006H    4  TOFILEEXTPTR         POINTER PARAMETER AUTOMATIC
                                       366  374  399  418

364  000AH    4  TOFILENAMEPTR        POINTER PARAMETER AUTOMATIC
                                       366  374  399  417

367  0002H    1  TOLINKSECT           BYTE
                                       422  426  436  437

367  0001H    1  TOLINKTRK            BYTE
                                       423  426

943  004AH    2  TOTALSECTORS         WORD
                                      1030 1031

 67  000AH    4  TPNTR                STRUCTURE AT
                                       521  523  565  567  840  841  848  849  851  852

167  0006H    1  TRACK                BYTE PARAMETER AUTOMATIC
                                       168  170

149  0008H    1  TRACK                BYTE PARAMETER AUTOMATIC
                                       150  163

132  0008H    1  TRACK                BYTE PARAMETER AUTOMATIC
                                       133  145

 57  0053H    1  TRACKNUMBER          BYTE
                                       202  216  223  230

942  2470H       TRANSFER             LABEL
                                       953  964  971  983  986

370  0EC0H       TRANSFER             LABEL
                                       373  408

 53  0001H    1  TRK                  BYTE MEMBER(LINK2)
                                       218  223  230  236  239  241  429  437  514  517  525

 52  0001H    1  TRK                  BYTE MEMBER(LINK1)
                                       347  352  428  435  552  561  920

 56             TRUE                 LITERALLY
                                       217  343  424  507  549  783  917

820  0095H    1  TYPE                 BYTE
                                       863  865  866

 43  0000H       USERIOGETCHAR        PROCEDURE EXTERNAL(16) STACK=0000H
                                       610  612

 13  0000H    2  W                    WORD PARAMETER
                                        14
```

```
  450  1155H   883  WRITE                    PROCEDURE STACK=0050H
                                               726

  723  1B43H    77  WRITEDRIVER              PROCEDURE PUBLIC STACK=0054H

   71  0016H     1  X0                       BYTE PARAMETER AUTOMATIC
                                                72   73

   95  0068H     1  X1                       BYTE
                                                98   99  100  101

   71  0014H     1  X1                       BYTE PARAMETER AUTOMATIC
                                                72   74

   71  0012H     1  X2                       BYTE PARAMETER AUTOMATIC
                                                72   75

   71  0010H     1  X3                       BYTE PARAMETER AUTOMATIC
                                                72   76

   71  000EH     1  X4                       BYTE PARAMETER AUTOMATIC
                                                72   77

   71  000CH     1  X5                       BYTE PARAMETER AUTOMATIC
                                                72   78

   71  000AH     1  X6                       BYTE PARAMETER AUTOMATIC
                                                72   79

   71  0008H     1  X7                       BYTE PARAMETER AUTOMATIC
                                                72   80

   71  0006H     1  X8                       BYTE PARAMETER AUTOMATIC
                                                72   81

   71  0004H     1  X9                       BYTE PARAMETER AUTOMATIC
                                                72   82


MODULE INFORMATION:

    CODE AREA SIZE     = 2722H  10018D
    CONSTANT AREA SIZE = 0000H      0D
    VARIABLE AREA SIZE = 00A5H    165D
    MAXIMUM STACK SIZE = 005CH     92D
    1362 LINES READ
    0 PROGRAM ERROR(S)

END OF PL/M-86 COMPILATION
```

III. DISK86: THE ASSEMBLY LANGUAGE SYSTEM INTERFACE

The critical nature of the time of the software interface to WD1791 Disk controller implied the absolute necessity that certain areas of the software interface be written in the 8086 Assembly Language. The original designers of this software/hardware interface were Lt Mike Baumgartner and Lt Dan McGrath. They originally tested the hardware software interface using a Z-80 based system running at 2MHz. The assembly language software was then converted from ASM80 source code to ASM86 source code using the 8086 utility program called CONV86. Some minor modification and enhancement followed this conversion with the final program being debugged by Captain J. Pollard.

The means of interface between the HOL programming in DOS86 and DISK86 is the IOPB (Input Output Parameter Block). The IOPB used is patterned after that used in the MDS-800 system to allow for the possible eventual applicaton of programs orginally written in 8080 code for the MDS-800. Table 5 shows the generic content of the IOPB as it relates to the USAFA/8086.

TABLE 5. IOPB DESCRIPTION FOR USAFA/8086

WORD	NAME	USAGE IN USAFA/8086
0	IOPB0	Channel Word. Normally=00H.
1	IOPB1	Disk Command Word. Drive # in bits 4 and 5. Operation Code in Bits 2, 1, and 0.

CODE	OPERATION
001	Seek
011	Restore
100	Read
101	Verify
110	Write
111	Write Deleted Data

Codes 000 and 010 are invalid.

2	IOPB2	Number of Sectors involved in this operation. (Normally=01H).
3	IOPB3	Track Number ($0 \leq to \leq 76_{10}$)
4	IOPB4	Sector Number ($1 \leq to \leq 26_{10}$)
5	IOPB5	Low and High halves of the starting address of any data transfer
6	IOPB6	
7	IOPB7	IOPB block #
8	IOPB8	Low and high halves of the successor IOPB if applicable (Channel word=04H)
9	IOPB9	

A high order flow diagram for DISK86 is shown in Figure 17.

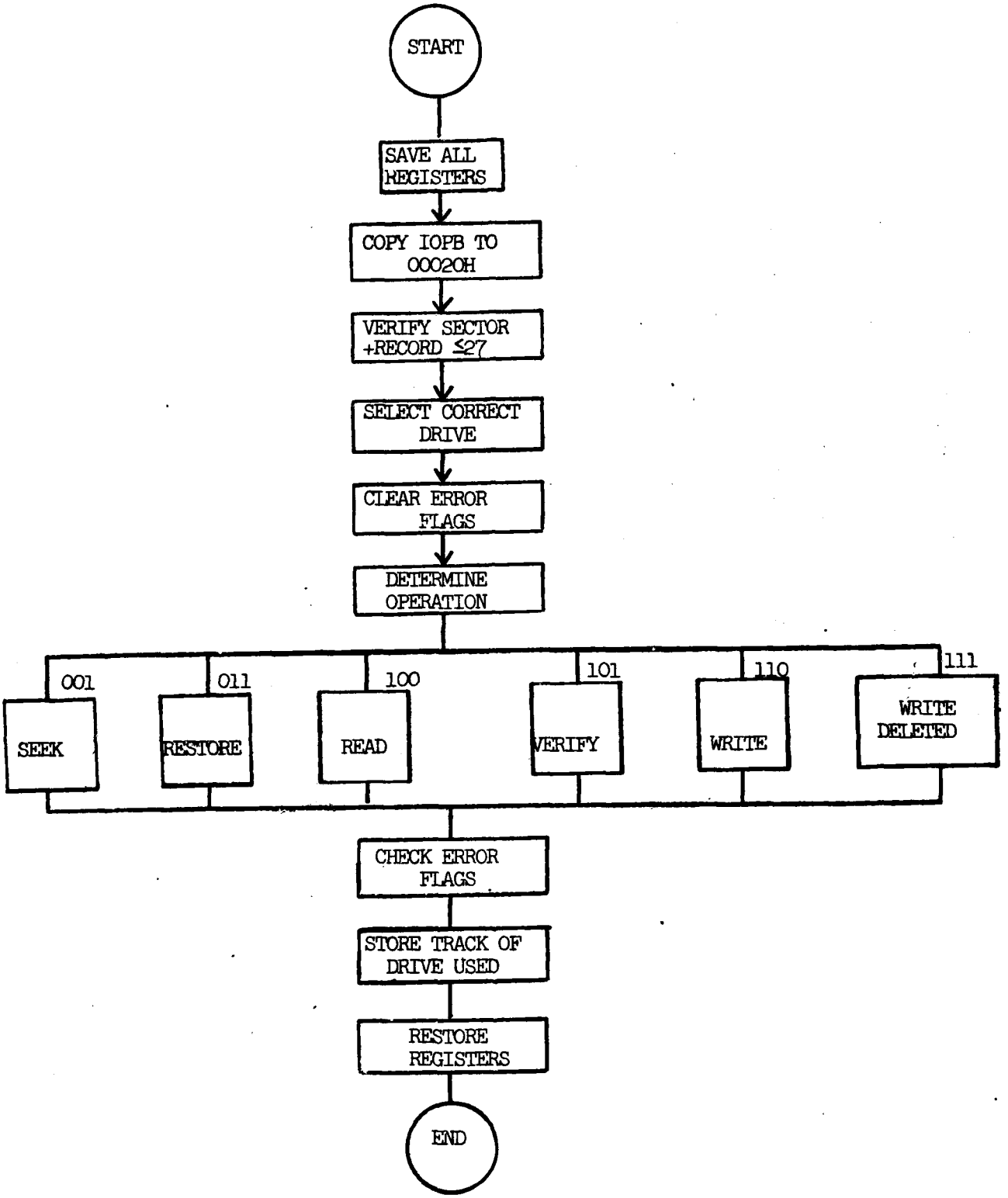


FIGURE 17. HIGH ORDER FLOW DIAGRAM FOR DISK86

The READ and WRITE routines contain extremely critical timing loops which should not be disturbed by any user without first considering the impact of his change. The DISK86 software sets error flags based conditions found during disk operations. These are delineated in Table 6.

TABLE 6. DISK ERROR WORD CODES

WORD	ERROR CODE	ERROR
FTERR(0002F)	01	Attempt to Read Deleted Data
	02	Sector Cannot be Written
	04	Seek Error Sects Not Found
	08	Bad CRC
	10	Data Lost During Reads Write
DERR(00035H)	08	Sector Address > 26_{10}
	10	Not Used
	20	Disk Write Protected
	40	Not Used
	80	Disk Not Ready

The DISK86 Module uses a large number of dedicated absolute memory locations These are indicated in Table 7.

TABLE 7. ABSOLUTE MEMORY ALLOCATION DISK86

LOCATION	NAME	USAGE
00020-00029	IOPB	A COPY OF THE USER IOPB
0002A	SWSV	DISK CONTROLLER STATUS
0002B	BCRC	#OF BAD CRC'S ENCOUNTERED
0002C	PRST	#OF TIMES CONTROLLER WAS RESET BY SOFTWARE
0002D	LOSTD	#OF TIMES DATA WAS LOST IN TRANSFER
0002E	PRNF	#OF TIMES A RECORD WAS NOT FOUND
0002F	FTERR	ERROR INDICATOR SEE TABLE 5
00030	DRNBR	CURRENT DRIVE NUMBER
00031	DR0TR	DRIVE 0'S CURRENT TRACK
00032	DR1TR	DRIVE 1'S CURRENT TRACK
00033	DR2TR	DRIVE 2'S CURRENT TRACK
00034	DR3TR	DRIVE 3'S CURRENT TRACK
00035	DERR	ERROR INDICATOR SEE TABLE 5
00036	IOPB	ADDRESS OF USER'S IOPB
00037	ADRS	
00038	BITBU	THE PROVERBIAL BIT BUCKET

A copy of the commented DISK86 software follows.

ISIS-II MCS-86 ASSEMBLER V1.0 ASSEMBLY OF MODULE DISK
OBJECT MODULE PLACED IN :F1:DISK.OBJ
ASSEMBLER INVOKED BY: ASM86 :F1:DISK.A86

```
LOC  OBJ                  LINE   SOURCE

                             1   DGROUP  GROUP   ASMCODE,CODE,CONST,DATA,STACK,MEMORY
                             2   DGROUP  GROUP   ASMCODE,CODE,CONST,DATA,STACK,MEMORY
                             3           ASSUME  DS:DGROUP,CS:CGROUP,SS:DGROUP
----                         4   CODE    SEGMENT WORD PUBLIC 'CODE'
----                         5   CODE    ENDS
----                         6   CONST   SEGMENT WORD PUBLIC 'CONST'
----                         7   CONST   ENDS
----                         8   DATA    SEGMENT WORD PUBLIC 'DATA'
----                         9   DATA    ENDS
----                        10   STACK   SEGMENT WORD STACK 'STACK'
0000                        11   STACK_BASE      LABEL   BYTE
----                        12   STACK   ENDS
----                        13   MEMORY  SEGMENT WORD MEMORY 'MEMORY'
0000                        14   MEMORY_ LABEL   BYTE
----                        15   MEMORY  ENDS
----                        16   ABS_0   SEGMENT BYTE
0000                        17   M       LABEL   BYTE
----                        18   ABS_0   ENDS
----                        19   ASMCODE SEGMENT BYTE
                            20    ;*** DISK *** PERFORMS ALL DISKETTE OPERATIONS EXCEPT FORMAT.
                            21    ;PARAMETERS  IOPB STARTING ADDRESS IN EXTERNAL IOPB
                            22    ;PROGRAMMER: DAN MCGRATH, MAY 1979
                            23    ;            MODIFIED JOE PULLARD, OCT 79
                            24              ;;; LABEL DEFINITIONS ;;;
  0020                      25   IOPB0  EQU  00020H   ;
  0021                      26   IOPB1  EQU  00021H   ;
  0022                      27   IOPB2  EQU  00022H   ;
  0023                      28   IOPB3  EQU  00023H   ; RESERVED MEMORY STORAGE
  0024                      29   IOPB4  EQU  00024H   ;  FOR INPUT/OUTPUT
  0025                      30   IOPB5  EQU  00025H   ;  PARAMETER BLOCK
  0026                      31   IOPB6  EQU  00026H   ;
  0027                      32   IOPB7  EQU  00027H   ;
  0028                      33   IOPB8  EQU  00028H   ;
  0029                      34   IOPB9  EQU  00029H   ;
  002A                      35   SWSV   EQU  0002AH   ; TEMP STORAGE OF STATUS WORD
  002B                      36   BCRC   EQU  0002BH   ; COUNT OF BAD REDUNDANCY CHECKS
  002C                      37   PRST   EQU  0002CH   ; COUNT OF CONTROLLER RESETS
  002D                      38   LOSTD  EQU  0002DH   ; NUMBER TIMES DATA LOST IN TRANSFER
  002E                      39   PRNF   EQU  0002EH   ; NUMBER TIMES RECORD NOT FOUND
  002F                      40   FTERR  EQU  0002FH   ; FATAL ERROR MESSAGE WORD
  0030                      41   DRNBR  EQU  00030H   ; NUMBER OF DRIVE CURRENTLY ENABLED
  0031                      42   DR0TR  EQU  00031H   ; TRACK LOCATION OF DRIVE 0
  0032                      43   DR1TR  EQU  00032H   ; TRACK LOCATION OF DRIVE 1
  0033                      44   DR2TR  EQU  00033H   ; TRACK LOCATION OF DRIVE 2
  0034                      45   DR3TR  EQU  00034H   ; TRACK LOCATION OF DRIVE 3
  0035                      46   DERR   EQU  00035H   ; DISK ERROR BYTE
  0036                      47   IOPBO   EQU     00036H              ; LOCATION 36 AND 37 CONTAINS IOPB
  0039                      48   BITEU   EQU     00039H              ; BIT BUCKET LOCATION
                            49                       ; I/O PORT DEFINITIONS
```

```
LOC  OBJ                  LINE   SOURCE

  00C8                      50   STARG  EQU 0C8H      ; CONTROLLER STATUS REGISTER
  00CA                      51   TRKRG  EQU 0CAH      ; CONTROLLER TRACK REGISTER
  00CC                      52   SECRG  EQU 0CCH      ; CONTROLLER SECTOR REGISTER
  00CE                      53   DATRG  EQU 0CEH      ; CONTROLLER DATA REGISTER
  00C8                      54   COMRG  EQU 0C8H      ; CONTROLLER COMMAND REGISTER
  00C4                      55   CHRST  EQU 0C4H      ; CONTROLLER RESET
  00C0                      56   DRSLE  EQU 0C0H      ; DRIVE SELECT PORT
  00C2                      57   DRHAK  EQU 0C2H      ; HANDSHAKE SIGNAL TO CONTROLLER
0000                        58   DISK   PROC   FAR    ;
                            59          PUBLIC DISK   ;
                            60          EXTRN IOPB:WORD
                            61          PUBLIC FTERR
                            62          PUBLIC DERR
0000 06                     63          PUSH ES
0001 1E                     64          PUSH DS
0002 50                     65          PUSH AX
0003 51                     66          PUSH CX
0004 52                     67          PUSH DX
0005 53                     68          PUSH BX
0006 56                     69          PUSH SI
0007 57                     70          PUSH DI
0008 BB0000                 71          MOV BX,0000H
000B 8EDB                   72          MOV DS,BX
000D 8EC3                   73          MOV ES,BX
000F 8B1E3600               74          MOV BX,DS.WORD PTR 36H
0013 8BCB                   75   ENTR1: MOV CX,BX
0015 268A870000      R      76          MOV AL,ES:[BX]
001A 2402                   77          AND   AL,02H          ; CHANNEL WORD
001C 7403                   78          JZ    SHORT L01       ;  IF ZERO, CONTINUE
001E E9F500                 79          JMP   SKIP            ;   ELSE, SKIP CURRENT IOPB
0021 BB2000                 80    L01:  MOV   BX,IOPB0        ;SET DESTINATION PTR
0024 B20A                   81          MOV   DL,0AH          ;SET COUNTER (SRC PTR=CX)
                            82                                ;SAVE IOPB IN RESERVED MEM
0026 87D9                   83   STPB:  XCHG  BX,CX           ;SWITCH POINTERS
0028 268A870000      R      84          MOV   AL,ES:[BX]        ; GET A BYTE FROM SRC IOPB
002D 87D9                   85          XCHG  BX,CX           ; SWITCH AGAIN
002F 2688870000      R      86          MOV   ES:[BX],AL        ; STORE IN DEST IOPB
0034 43                     87          INC   BX              ; INCR DEST PTR
0035 41                     88          INC   CX              ; INCR SRC PTR
0036 FECA                   89          DEC   DL              ; DECR COUNTER
0038 75EC                   90          JNZ     STPB:   CHECK FOR ADDR ERROR
003A 268A1E2200      R      91          MOV   BL,ES:[IOPB2]     ; ADD NUMBER OF RECORDS
003F 268A02400       R      92          MOV   AL,ES:[IOPB4]     ; AND SECTOR ADDR
0043 02C3                   93          ADD   AL,BL           ;
0045 3C1B                   94          CMP   AL,27D          ; CHECK FOR SUM>26
0047 7E07                   95          JLE   SHORT SELDR     ;
0049 26C606350008    R      96          MOV   ES:[DERR],08H         ; IF >26, AL=ADDRESS
004F CB                     97          RET                   ;  ERROR, ABORT.
                            98                                ;SELECT DRIVE AND SET
                            99                                ; UP TRACK REGISTER
0050 26A02100        R     100   SELDR: MOV   AL,ES:[IOPB1]     ;GET DISKETTE INSTR
0054 2430                  101          AND   AL,30H           ; AND MASK FOR DRV NBR
0056 26A23000        R     102          MOV   ES:[DRNBR],AL    ; SAVE DRV NBR IN MEM
005A 3C00                  103          CMP   AL,00H          ; DRIVE 0?
005C 7599                  104          JNZ   SHORT DR10      ; NO, JUMP
```

```
LOC  OBJ                  LINE  SOURCE

005E B001                  105          MOV   AL,01H          ; YES, AL=DRIVE 0 SEL MASK
0060 268A1E3100      R     106          MOV   BL,ES:MLDR0TR]    ; BL=LAST TRACK ADDR
0065 EB21                  107          JMP   SHORT DRSEL     ;
0067 3C10                  108   DR10:  CMP   AL,10H          ;DRIVE 1?
0069 7509                  109          JNZ   SHORT DR20      ; NO, JUMP
006B B002                  110          MOV   AL,02H          ; AL=DRIVE 1 MASK
006D 268A1E3200      R     111          MOV   BL,ES:MLDR1TR]    ; BL=LAST TRACK ADDR
0072 EB14                  112          JMP   SHORT DRSEL     ;
0074 3C20                  113   DR20:  CMP   AL,20H          ; CHECK FOR DRIVE 2
0076 7509                  114          JNZ   SHORT DR30      ; NO, JUMP
0078 B004                  115          MOV   AL,04H          ; YES, AL=DRV 2 MASK
007A 268A1E3300      R     116          MOV   BL,ES:MLDR2TR]    ; BL=LAST TRACK ADDR
007F EB07                  117          JMP   SHORT DRSEL     ;
0081 B003                  118   DR30:  MOV   AL,08H          ; ASSUME DRV 3, AL=MASK
0083 268A1E3400      R     119          MOV   BL,ES:MLDR3TR]    ; BL=LAST TRACK, DRV 3
0088 E6C0                  120   DRSEL: OUT   DRSLE,AL        ;OUTPUT DRV SELECT MASK
008A 8AC3                  121          MOV   AL,BL
008C F6D0                  122          NOT   AL              ;###
008E E6CA                  123          OUT   TRKRG,AL        ;LOAD TRACK REGISTER
                           124                                ;CHECK FOR DRIVE READY
0090 E4C8                  125          IN      AL,STARG      ; READ STATUS WORD
0092 F6D0                  126          NOT     AL            ; ###
0094 2480                  127          AND     AL,80H        ; CHECK READY BIT
0096 7407                  128          JZ      SHORT CLRER   ; BRANCH IF READY
0098 26C606250080    R     129          MOV     ES:MLDERR],80H       ; ELSE, AL=ERROR INDICATION
009E CB                    130          RET                   ; ABORT.
                           131                                ;CLEAR ERROR RECORDS
009F B206                  132   CLRER: MOV   DL,06H          ; ZERO OUT THE
00A1 BB2A00                133          MOV   BX,SWSV         ; SIX RESERVED MEM
00A4 B000                  134          MOV   AL,00H          ; LOCATIONS
00A6 2688870000      R     135   CLR:   MOV   ES:MLBX],AL       ; FROM SWSV
00AB 43                    136          INC   BX              ; TO FTERR
00AC FECA                  137          DEC   DL              .
00AE 75F6                  138          JNZ   CLR             ;
00B0 26A23500        R     139          MOV     ES:MLDERR],AL ; CLEAR DERR
                           140                                ;DETERMINE OPERATION AND VECTOR
                           141                                ; TO SUBROUTINE
00B4 26A02100        R     142   OPER:  MOV     AL,ES:MLIOPB1]        ;DETERMINE OPERATION::
00B8 2407                  143          AND     AL,07H                ; MASK DISK INSTR FOR OP CODE
00BA 2C01                  144          SUB     AL,01H                ; OP CODE =001? (SEEK)
00BC 7505                  145          JNZ     SHORT OP3             ; NO, CHECK 011
00BE E8AA00                146          CALL    SEEK                  ; YES, CALL SEEK ROUTINE
00C1 EB4B                  147          JMP     SHORT THRU            ;  THEN GO TO END
00C3 2C02                  148   OP3:   SUB     AL,02H                ; OP CODE =011? (RESTORE)
00C5 7514                  149          JNZ     SHORT OP4             ; NO, CHECK 100
00C7 B008                  150          MOV     AL,08H                ; YES, LOAD COMMAND REG
00C9 F6D0                  151          NOT     AL                    ;###
00CB E6C8                  152          OUT     COMRG,AL              ;  WITH RESTORE INSTR
00CD E4C8                  153    R1:   IN      AL,STARG              ; WAIT
00CF 2401                  154          AND     AL,01H                ;  FOR BUSY
00D1 75FA                  155          JNZ     R1                    ;   BIT TO GO HIGH
00D3 E4C8                  156    R2.   IN      AL,STARG              ;    AND
00D5 2401                  157          AND     AL,01H                ;     BACK
00D7 74FA                  158          JZ      R2                    ;      LOW
00D9 EB33                  159          JMP     SHORT THRU            ;       AND GO TO END
```

```
LOC  OBJ              LINE  SOURCE

00D8 2C01             160   OP4:   SUB   AL,01H              ; OP CODE =100? (READ)
00DA 7505             161          JNZ   SHORT OP5            ; NO, CHECK 101
00DC E81201           162          CALL  READ                 ; YES, CALL READ ROUTINE
00DF EB2A             163          JMP   SHORT THRU           ;  THEN GO TO END
00E1 2C01             164   OP5:   SUB   AL,01H              ; OP CODE =101? (VERIFY CRC)
00E3 7505             165          JNZ   SHORT OP6            ; NO, CHECK 110
00E5 E82503           166          CALL  VERFY                ; YES, CALL VERIFY ROUTINE
00E8 EB21             167          JMP   SHORT THRU           ;  THEN GO TO END
00EA 2C01             168   OP6:   SUB   AL,01H              ; OP CODE =110? (WRITE)
00EC 750D             169          JNZ   SHORT OP7            ; NO, CHECK 111
00EE E4C8             170   WR:    IN    AL,STARG             ; CHECK WRITE PROTECT
00F0 F6D0             171          NOT   AL                   ;###
00F2 2440             172          AND   AL,40H               ; WPROT=1, DO NOT WRITE
00F4 750E             173          JNZ   SHORT NOWR           ; WPROT=0, CALL WRITE
00F6 E80E02           174          CALL  WRITE                ;  THEN GO TO END
00F9 EB10             175          JMP   SHORT THRU           ;
00FB 2C01             176   OP7:   SUB   AL,01H              ; OP CODE=111? (WR DEL DATA)
00FD 750C             177          JNZ   SHORT THRU           ; NO, GO TO END
00FF B6A1             178          MOV   DH,0A1H              ; YES, D=WRITE DELETED INSTR
0101 E80302           179          JMP   WRITE                ; GO AND CALL WRITE
                      180
0104 26C606350020  R  181   NOWR:  MOV   ES:MCDERR],20H              ;AL=WRT PROT INDIC
0109 CB               182          RET                        ; ABORT.
                      183                              ;COMMON RETURN PT FOR ALL OP'S
010A 26A02F00      R  184   THRU:  MOV   AL,ES:MCFTERR]     ; GET FATAL ERROR MESSAGE
010E 24FF             185          AND   AL,0FFH        ;  CHECK FOR ERRORS
0110 7508             186          JNZ   SHORT CONC     ;
                      187                               ; AT THIS POINT, IT IS ASSUMED THAT
                      188                               ; A SUCCESSFUL OPERATION HAS OCURRED
0112 26A02000      R  189   SKIP:  MOV   AL,ES:MCIOPB0]    ;GET CHANNEL WORD AND
0116 2404             190          AND   AL,04H         ; CHECK SUCCESSOR BIT
0118 753F             191          JNZ   SHORT REPT     ; IF ZERO, CONCLUDE OPERATION:
011A 26A03000      R  192   CONC.  MOV   AL,ES:MCDRNBR]   ; RETRIEVE DRIVE NUMBER
011E 3C00             193          CMP   AL,00H         ; DRIVE 0?
0120 750A             194          JNZ   SHORT DRV1     ; NO, JUMP
0122 E4CA             195          IN    AL,TRKRG       ; READ TRACK REGISTER
0124 F6D0             196          NOT   AL             ; ###
0126 26A23100      R  197          MOV   ES:MCDR0TR],AL     ; SAVE
012A EB24             198          JMP   SHORT FINSH
012C 3C10             199   DRV1:  CMP   AL,10H         ; DRIVE 1?
012E 750A             200          JNZ   SHORT DRV2     ; NO, JUMP
0130 E4CA             201          IN    AL,TRKRG       ; ELSE READ TRACK REG
0132 F6D0             202          NOT   AL             ; ###
0134 26A23200      R  203          MOV   ES:MCDR1TR],AL     ; SAVE
0138 EB16             204          JMP   SHORT FINSH
013A 3C20             205   DRV2.  CMP   AL,20H         ; DRIVE 2?
013C 750A             206          JNZ   SHORT DRV3     ; NO, JUMP
013E E4CA             207          IN    AL,TRKRG       ; ELSE, READ TRACK
0140 F6D0             208          NOT   AL             ; ###
0142 26A23300      R  209          MOV   ES:MCDR2TR],AL     , SAVE
0146 EB08             210          JMP   SHORT FINSH
0148 E4CA             211   DRV3:  IN    AL,TRKRG       ; ASSUME DRIVE 3
014A F6D0             212          NOT   AL             ; ###
014C 26A23400      R  213          MOV   ES:MCDR3TR],AL     ; SAVE TRACK NUMBER
0150 5F               214   FINSH: POP DI
```

```
LOC  OBJ                  LINE   SOURCE

0155 5E                    215          POP SI
0156 5B                    216          POP BX
0157 5A                    217          POP DX
0158 59                    218          POP CX
0159 58                    219          POP AX
015A 1F                    220          POP DS
015B 07                    221          POP ES
015C CB                    222          RET
                           223  DISK    ENDP,
                           224                  ;CONCLUDE
015D BB4006                225   REPT:  MOV   BX,0640H       ;
0160 E82903                226          CALL  DELAY          ;DELAY 16 MILLISECONDS
0163 268B1E2800      R     227          MOV    BX,WORD PTR ES:[IOPB8] ; LOAD BX WITH NEXT IOPB ADDRESS
0168 E9A8FE                228          JMP   ENTR1          ;PERFORM NEXT OPERATION
                           229
                           230                  ;SUBROUTINE SEEK :::
                           231                  ; MOVES DISK HEAD TO TRACK SPECIFIED
                           232                  ; IN IOPB[3]
016B                       233  SEEK           PROC    NEAR;
016B 26A03000        R     234                  MOV     AL,ES:[DRNBR];GET CURRENT DRIVE NUMBER
016F D0E8                  235                  SHR     AL,1; ALIGN ADDRESS
0171 D0E8                  236                  SHR     AL,1;
0173 D0E8                  237                  SHR     AL,1;
0175 D0E8                  238                  SHR     AL,1;
0177 FEC0                  239                  INC     AL;     ADD ONE FOR HARDWARE
 179 0430                  240                  ADD     AL,DRNBR;       ADD OFFSET TO GET CURRENT TRACK ADDR
017B B700                  241                  MOV     BH,00H
017D 8AD8                  242                  MOV     BL,AL
017F 268A870000      R     243                  MOV     AL,ES:[BX];    LOAD CURRENT TRACK
0184 F6D0                  244                  NOT     AL
0186 E6CA                  245                  OUT     TRKRG,AL;       UPDATE TRACK REGISTER
0188 F6D0                  246                  NOT     AL;     RETURN TO POSITIVE LOGIC
018A 263A062300      R     247                  CMP     AL,ES:[IOPB3]; COMPARE TO DESIRED TRACK
018F 742A                  248                  JZ      EXITSK
0191 26A02300        R     249          MOV     AL,ES:[IOPB3]          ;GET TRACK ADDR AND
0195 F6D0                  250          NOT     AL                     ;###
0197 E6CE                  251          OUT     DATRG,AL               ;LOAD INTO DATA REG
0199 B80100                252          MOV     BX,0001H
019C E8EE02                253          CALL    DELAY                  ; WAIT 10 MICROSECONDS
019F B01F                  254          MOV     AL,1FH                 ;LOAD SEEK INSTR
01A1 F6D0                  255          NOT     AL                     ;###
01A3 E6C8                  256          OUT     COMRG,AL               ; INTO COMMAND REG
01A5 B80100                257          MOV     BX,0001H     ; SET DELAY TIME
01A8 E8E202                258          CALL    DELAY        ; WAIT 10 MICROSECS
                           259                               ;CHECK FOR DONE
01AB E4C8                  260  ENDSK:  IN      AL,STARG       ; READ STATUS REGISTER
01AD F6D0                  261          NOT     AL             ; ###
01AF 2401                  262          AND     AL,01H         ; CHECK BUSY BIT
01B1 75F8                  263          JNZ     ENDSK          ; UNTIL ZERO
01B3 E4C9                  264          IN      AL,STARG               ;READ STATUS REG
01B5 F6D0                  265          NOT     AL                     ;###
 1B7 2410                  266          AND     AL,10H                 ;CHECK SEEK ERROR BIT
01B9 7501                  267          JNZ     SHORT L_5
01BB C3                    268  EXITSK. RET
01BC                       269  L_5:                                   ;IF ZERO, RETURN, ELSE
```

```
LOC  OBJ                LINE   SOURCE

01BC BB2C00             270            MOV     BX,PRST                 ;IN CASE OF SEEK ERROR
01BF 268A870000      R  271            MOV     AL,ES:[BX]                 ; CHECK RESET RECORD
01C4 3C01               272            CMP     AL,01H                  ; IF 1 PRIOR RESET,
01C6 7411               273            JZ      SHORT REST              ;  JUMP
01C8 26FE870000      R  274            INC     ES:[BX]                    ; ELSE, UPDATE RECORD
01CD B0FF               275            MOV     AL,0FFH                 ; SET MASK,
01CF E6C4               276            OUT     CHRST,AL                ; RESET CONTROLLER,
01D1 B80500             277            MOV     BX,00005H               ; SET DELAY TIME
01D4 E89682             278              CALL    DELAY                   ; DELAY 50 MICRO
01D7 EB92               279            JMP     SEEK                    ; AND TRY AGAIN
01D9 BB2E00             280    REST:   MOV     BX,PRNF                 ; CHECK REC NOT FOUND
01DC 268A870000      R  281            MOV     AL,ES:[BX]                 ;
01E1 3C02               282            CMP     AL,02H                  ; 2 ERRORS AFTER RESET?
01E3 7408               283            JZ      SHORT FATAL             ; YES, JUMP
01E5 26FE870000      R  284            INC     ES:[BX]                    ; NO, UPDATE RECORD,
01EA E97EFF             285            JMP     SEEK                    ;  AND TRY AGAIN
01ED 26C6062F0004    R  286    FATAL:  MOV     ES:[LFTERR],04H                 ;FATAL ERROR: GET F.E
                        287                                            ; MESSG, SET SEEK ERROR,
01F3 C3                 288            RET                             ; AND RETURN TO DISK
                        289    SEEK    ENDP;
                        290
                        291                                    ;SUBROUTINE READ :::
01F4                    292    READ    PROC    NEAR;
01F4 E874FF             293            CALL    SEEK                    ;SEEK TRACK
01F7 BB2200             294            MOV     BX,IOPB2                ;GET #RECORDS
01FA 268A0F0000      R  295            MOV     CH,ES:[BX]                 ; AND LOAD INTO COUNT CH
01FF 43                 296            INC     BX                      ; GET SECTOR
0200 43                 297            INC     BX                      ; ADDRESS AND
0201 268A870000      R  298            MOV     AL,ES:[BX]              ; LOAD INTO
0206 F6D0               299            NOT     AL                      ;###
0208 E6CC               300            OUT     SECRG,AL                ; SECTOR REGISTER
020A B180               301    FLAG1:  MOV     CL,80H                  ;COUNT C = #BYTES/SECT
020C B084               302            MOV     AL,84H                  ;LOAD COMMAND REG
020E F6D0               303            NOT     AL                      ;###
                        304                                            ;MEM PTR = DEST ADDR
0210 268B1E2500      R  305            MOV     BX,WORD PTR ES:[IOPB5]  ;
0215 B2C8               306            MOV     DL,STARG                ; DL=STATUS PORT
0217 B402               307            MOV     AH,02H                  ; AH=STATUS MASK
0219 E6C8               308            OUT     COMRG,AL                ; OUTPUT TO CONTROLLER COMMAND
021B EC                 309    RLOOP:  IN      AL,DX                   ; READ STATUS
021C 22C4               310            AND     AL,AH                   ; UNTIL DRQ
021E 75FB               311            JNZ     RLOOP                     ; ACTIVE
0220 E4CE               312            IN      AL,DATRG                ; READ A BYTE
0222 F6D0               313            NOT     AL                      ; ###
0224 88870000        R  314            MOV     DS:[BX],AL                 ; STORE AT DEST MEM
0228 43                 315            INC     BX                      ; UPDATE DEST PTR
0229 FEC9               316            DEC     CL                      ; DECR COUNTER
022B 75EE               317            JNZ     RLOOP                   ; CONTINUE UNTIL CL=0
                        318                                    ;CHECK FOR DONE
022D E4C8               319    ENDRD:  IN      AL,STARG        ; READ STATUS
022F F6D0               320            NOT     AL              ; ###
0231 2401               321            AND     AL,01H          ; CHECK BUSY BIT
0233 75F8               322            JNZ     ENDRD           ; UNTIL ZERO
0235 E4C8               323            IN      AL,STARG                ;READ STATUS REG.
0237 F6D0               324            NOT     AL                      ;###
```

```
LOC  OBJ                  LINE   SOURCE

0239 26A22A00        R     325            MOV     ES:MCSWSV],AL           ;SAVE STATUS WORD
023D 2408                  326            AND     AL,08H                  ; CHECK CRC BIT
023F 741A                  327            JZ      SHORT RNF               ;  IF OK, CHECK RNF
0241 BB2B00                328    CRCC:    MOV      BX,BCRC                 ;GET CRC RECORD
0244 268A870000      R     329            MOV     AL,ES:MCBX]               ; CHECK FOR MORE THAN
0249 2C03                  330            SUB     AL,03H                  ;  THREE TIMES
024B 7407                  331            JZ      SHORT CRE               ;  JUMP IF 3
024D 26FE870000      R     332            INC     ES:MCBX]                  ; ELSE UPDATE RECORD
0252 EBB6                  333            JMP     FLAG1                   ;  AND TRY ANOTHER READ
0254 26C6862F0008    R     334     CRE:   MOV     ES:MFTERR],08H                  ; 3 BAD CRC'S:
                           335                                            ; LOAD FATAL MESSAGE
025A C3                    336            RET                             ;ABORT
025B 26A02A00        R     337     RNF:   MOV     AL,ES:MCSWSV]             ; GET STATUS
025F 2410                  338            AND     AL,10H                  ; CHECK RNF BIT
0261 743C                  339            JZ      SHORT CHKLD             ; JUMP IF O.K.
0263 BB2E00                340            MOV     BX,PRNF                 ; ELSE GET RNF RECORD
0266 268A870000      R     341            MOV     AL,ES:MCBX]             ; HAS OPERATION
026B 2C02                  342            SUB     AL,02H                  ; FAILED TWICE?
026D 7408                  343            JZ      SHORT RESTR             ; YES, TRY RESET
026F 26FE870000      R     344            INC     ES:MCBX]                      ; NO, INCR RNF REC
0274 E97DFF                345            JMP     READ                    ; AND TRY AGAIN
0277 BB2C00                346    RESTR:  MOV     BX,PRST                 ;RESET: CHECK RESET
027A 268A870000      R     347            MOV     AL,ES:MCBX]             ; RECORD.  IF NOT ZERO,
027F 2C01                  348            SUB     AL,01H                  ; SET FATAL ERROR
0281 7412                  349            JZ      SHORT FATE              ; MESSAGE,OTHERWISE
0283 26FE870000      R     350            INC     ES:MCBX]                      ; INCR RESET RECORD
0288 F6D0                  351            NOT     AL                      ;###
028A E6C4                  352            OUT     CHRST,AL                ; RESET CONTROLLER
028C BB0500                353            MOV     BX,0005H                ; SET TIME
028F E8F001                354            CALL    DELAY                   ; DELAY 50 MICROSECONDS
0292 E95FFF                355            JMP     READ                    ; AND TRY AGAIN
0295 BB2F00                356    FATE:   MOV     BX,FTERR                ; GET FATAL MESSG BYTE
0298 26C6870000FF    R     357            MOV     ES:MCBX],0FFH           ; LOAD RNF MESSG AND
029E C3                    358            RET                             ; RETURN TO DISK
029F 26A02A00        R     359    CHKLD:  MOV     AL,ES:MCSWSV]           ;CHECK FOR LOST DATA:
02A3 2404                  360            AND     AL,04H                  ; CHK L.D. BIT
02A5 741E                  361            JZ      SHORT CHKDD             ; NO L.D., CHECK DEL. DATA
02A7 BB2D00                362            MOV     BX,LOSTD                ; L.D., GET L.D. RECORD
02AA 268A870000      R     363            MOV     AL,ES:MCBX]             ; HAS READ FAILED TWICE
02AF 2C02                  364            SUB     AL,02H                  ; DUE TO LOST DATA?
02B1 780A                  365            JS      SHORT LDLT              ; NO, JUMP
02B3 BB2F00                366            MOV     BX,FTERR                ; YES, LOAD FATAL MESSG
02B6 26C687000010    R     367            MOV     ES:MCBX],10H            ;  LOST DATA MESSG.
02BC C3                    368            RET                             ;  AND RETURN TO DISK
02BD 26FE870000      R     369    LDLT:   INC     ES:MCBX]                      ; INCR LOST DATA REC.
02C2 E945FF                370            JMP     FLAG1                   ; AND TRY AGAIN
02C5 26A02A00        R     371    CHKDD:  MOV     AL,ES:MCSWSV]           ;CHECK FOR DEL. DATA:
02C9 2420                  372            AND     AL,20H                  ; CHECK REC TYPE BIT
02CB 740A                  373            JZ      SHORT REP_              ; IF TYPE NORMAL, JUMP
02CD BB2F00                374            MOV     BX,FTERR                ; TYPE=DD, LOAD FATAL ERROR
02D0 26C687000001    R     375            MOV     ES:MCBX],01H            ; REC. WITH DD MESSG
02D6 C3                    376            RET                             ; AND RETURN
02D7 E4CC                  377    REP_:   IN      AL,SECRG                ;NO ERRORS: GET SECTOR REG
02D9 F6D0                  378            NOT     AL                      ;###
02DB FEC0                  379            INC     AL                      ; INCR SECTOR
```

```
LOC  OBJ                LINE   SOURCE

02DD F6D0                380           NOT     AL                      ;###
02DF E6CC                381           OUT     SECRG,AL                ;  AND RELOAD
02E1 FECD                382           DEC     CH                      ; DECR #RECORDS
02E3 7501                383           JNZ     SHORT L_6
02E5 C3                  384           RET
02E6                     385   L_6:                                    ; RETURN IF DONE, ELSE
02E6 268B1E2500      R   386           MOV     BX,WORD PTR ES:[IOPB5] ;UPDATE
02EB B98000              387           MOV     DX,80H                  ; DESTINATION
02EE 03DA                388           ADD     BX,DX
02F0 D1DE                389           RCR     SI,1
02F2 D1D6                390           RCL     SI,1                    ;  ADDRESS BY
02F4 26891E2500      R   391           MOV     WORD PTR ES:[IOPB5],BX ;       128 BYTES
02F9 B000                392           MOV     AL,00H                  ;CLEAR
02FB 26A22C00        R   393           MOV     ES:[PRST],AL            ; NON-FATAL
02FF 26A22D00        R   394           MOV     ES:[LOSTD],AL           ;  ERROR
0303 26A22E00        R   395           MOV     ES:[PRNF],AL            ;       RECORDS
0307 E900FF              396           JMP     FLAG1                   ; READ NEXT SECTOR
                         397   READ    ENDP;
                         398
                         399                                   ;SUBROUTINE WRITE :::
030A                     400   WRITE   PROC    NEAR;
030A E85EFE              401           CALL    SEEK                    ;MOVE TO DESIRED TRACK
030D BB2200              402           MOV     BX,IOPB2                ;GET NUMBER OF RECORDS
0310 268A0F0000      R   403           MOV     CH,ES:[BX]              ;COUNT B=#RECORDS
0315 43                  404           INC     BX
0316 43                  405           INC     BX
0317 268A070000      R   406           MOV     AL,ES:[BX]              ;LOAD SECTOR ADDR
031C F6D0                407           NOT     AL                      ;###
031E E6CC                408           OUT     SECRG,AL                ; INTO SECTOR REG
0320 B180                409   FLAGW:  MOV     CL,80H                  ;COUNTER C=128
0322 B0A4                410           MOV     AL,0A4H                 ;LOAD COMMAND REG WITH
0324 F6D0                411           NOT     AL                      ;###
                         412                                           ; WRITE OR WR DEL DATA INSTR
0326 268B362500      R   413           MOV     SI,WORD PTR ES:[IOPB5]   ; MEM PTR = SRC ADDR
032B B2C8                414           MOV     DL,STARG                ; DL=STATUS PORT
032D B402                415           MOV     AH,02H                  ; AH=MASK
032F E6C8                416           OUT     COMRG,AL                ; COMMAND TO CONTROLLER
0331 8A14                417   TLOOP:  MOV     DL,DS:[SI]              ; PRIME DATA REGISTER
0333 F6D2                418           NOT     DL
0335 E4C8                419   WLOOP:  IN      AL,STARG                ; READ STATUS
0337 22C4                420           AND     AL,AH                   ; UNTIL DRQ
0339 75FA                421           JNZ     WLOOP                   ; ACTIVE
033B 8AC2                422           MOV     AL,DL                   ; GET A BYTE FROM SRC MEM
033D E6CE                423           OUT     DATRG,AL                ; WRITE TO DISKETTE
033F 46                  424           INC     SI                      ; UPDATE PTR
0340 FEC9                425           DEC     CL                      ; DECREMENT COUNTER
0342 75ED                426           JNZ     TLOOP                   ; CONT. UNTIL COUNT=0
                         427                                           ;CHECK FOR DONE
0344 E4C8                428   ENDWR:  IN      AL,STARG                ; READ STATUS
0346 F6D0                429           NOT     AL                      ; ###
0348 2401                430           AND     AL,01H                  ; CHECK BUSY BIT
034A 75F8                431           JNZ     ENDWR                   ; UNTIL ZERO
034C E4C9                432           IN      AL,STARG                ;READ STATUS REG
034E F6D0                433           NOT     AL                      ;###
0350 26A22A00        R   434           MOV     ES:[SWSV],AL            ;CHECK FOR ERRORS.
```

```
LOC  OBJ                LINE  SOURCE

0354 2410               435           AND     AL,10H                  ; SAVE SW,CHECK RNF
0356 743A               436           JZ      SHORT CHLD              ; IF RNF=0, CHECK LOST DATA
0358 BB2E00             437   RECNF:  MOV     BX,PRNF                 ;IF RNF=1, GET RNF
035B 268A870000     R   438           MOV     AL,ES:[BX]              ; RECORD
0360 2C02               439           SUB     AL,02H                  ; TWO FAILURES?
0362 7407               440           JZ      SHORT RSET              ; YES, TRY RESET
0364 26FE870000     R   441           INC     ES:[BX]                         ; NO, INCR RNF REC
0369 EB9F               442           JMP     WRITE                   ; AND TRY AGAIN
036B BB2C00             443   RSET:   MOV     BX,PRST                 ;GET RESET REC
036E 268A870000     R   444           MOV     AL,ES:[BX]              ; IF NOT 0, SET
0373 2C00               445           SUB     AL,00H                  ; FATAL MESSG,
0375 7511               446           JNZ     SHORT FTAL              ; OTHERWISE,
0377 26FE870000     R   447           INC     ES:[BX]                         ; INCR RESET REC
037C F6D0               448           NOT     AL                      ;###
037E E6C4               449           OUT     CHRST,AL                ; RESET CONTROLLER
0380 BB0500             450           MOV     BX,0005H                ; SET TIME
0383 E80701             451           CALL    DELAY                   ; DELAY 50 MICROSECONDS
0386 EB82               452           JMP     WRITE                   ; AND TRY AGAIN
0388 BB2F00             453   FTAL:   MOV     BX,FTERR                ;GET FATAL MESSG BYTE
038B 26C6870000FF   R   454           MOV     ES:[BX],0FFH            ; SET RNF MESSG
0391 C3                 455           RET                             ; AND RETURN
0392 26A02A00       R   456   CHLD:   MOV     AL,ES:[SWSV]            ;CHECK LOST DATA:
0396 2404               457           AND     AL,04H                  ; CHECK STATUS WORD
0398 741A               458           JZ      SHORT VERIF             ; NO L.D.,VERIFY WRITE
039A 26A02D00       R   459           MOV     AL,ES:[LOSTD]           ; ELSE, GET L.D. REC
039E 2C02               460           SUB     AL,02H                  ; FAILED TWICE?
03A0 740A               461           JZ      SHORT LSTD              ; NO,JUMP,ELSE
03A2 BB2F00             462           MOV     BX,FTERR                ; LOAD FATAL MESSG
03A5 26C687000010   R   463           MOV     ES:[BX],10H             ; REC WITH L.D. INDIC
03AB C3                 464           RET                             ; AND RETURN TO DISK
03AC 26FE870000     R   465   LSTD:   INC     ES:[BX]                         ; INCR LOST DATA REC
03B1 E96CFF             466           JMP     FLAGW                   ; AND TRY AGAIN
03B4 EB2790             467   VERIF.  JMP     NXTR                    ;REPLACE CALL VRCRC JJP
03B7 26A02F00       R   468           MOV     AL,ES:[FTERR]           ; BY VERIFYING CRC
03BB 2402               469           AND     AL,02H                  ; FTERR=CRC MESSG?
03BD 741E               470           JZ      SHORT NXTR              ; NO, WRITE NEXT REC
03BF BB2B00             471           MOV     BX,BCRC                 ; ELSE, GET BAD CRC REC
03C2 268A870000     R   472           MOV     AL,ES:[BX]              ; AND CHECK FOR
03C7 2C06               473           SUB     AL,06H                  ; SIX FAILURES
03C9 7408               474           JZ      SHORT FCRC              ; FOR LESS THAN
03CB 26FE870000     R   475           INC     ES:[BX]                         ; UPDATE REC
03D0 E94DFF             476           JMP     FLAGW                   ; WRITE SECTOR OVER
03D3 BB2F00             477   FCRC:   MOV     BX,FTERR                ; ELSE, SET FATAL
03D6 26C687000002   R   478           MOV     ES:[BX],02H             ; ERROR MESSG AND
03DC C3                 479           RET                             ; RETURN TO DISK
03DD E4CC               480   NXTR:   IN      AL,SECRG                ;WRITE NEXT RECORD::
03DF F6D0               481           NOT     AL                      ;###
03E1 FEC0               482           INC     AL                      ; GET SECTOR, INCR,
03E3 F6D0               483           NOT     AL                      ;###
03E5 E6CC               484           OUT     SECRG,AL                ; AND RELOAD
03E7 FECD               485           DEC     CH                      ; DECREMENT SECTOR COUNT
03E9 7501               486           JNZ     SHORT L_7
03EB C3                 487           RET
03EC                    488   L_7:                                    ; RETURN IF DONE, ELSE
03EC 26881E2500     R   489           MOV     BX,WORD PTR ES:[IOPB5] ;UPDATE
```

```
LOC  OBJ              LINE  SOURCE

03F1 BA8000           490         MOV   DX,80H              ; DESTINATION
03F4 03DA             491         ADD   BX,DX
03F6 D1DE             492         RCR   SI,1
03F8 D1D6             493         RCL   SI,1               ;  ADDRESS BY
03FA 26891E2500   R   494         MOV   WORD PTR ES:[IOPB5],BX ;       128 BYTES
03FF B000             495         MOV   AL,00H             ;CLEAR
0401 26A22C00     R   496         MOV   ES:[PRST],AL       ; NON-FATAL
0405 26A22D00     R   497         MOV   ES:[LOSTD],AL      ;  ERROR
0409 26A22E00     R   498         MOV   ES:[PRNF],AL       ;       RECORDS
040D E910FF           499         JMP   FLAGW              ; WRITE NEXT SECTOR
                      500   WRITE ENDP;
                      501
                      502                        ;SUBROUTINE VERFY :::
0410                  503   VERFY PROC  NEAR;
0410 E858FD           504         CALL  SEEK               ;MOVE TO DESIRED TRACK
0413 BB2200           505         MOV   BX,IOPB2           ; B COUNT=# RECORDS
0416 268A8F0000   R   506         MOV   CH,ES:[BX]         ;
041B 26A02400     R   507         MOV   AL,ES:[IOPB4]      ;LOAD SECTOR REG
041F F6D0             508         NOT   AL                 ;###
0421 E6CC             509         OUT   SECRG,AL           ; WITH SECTOR ADDR
0423 E81E00           510   VER:  CALL  VRCRC              ;VERIFY SECTOR
0426 26A02F00     R   511         MOV   AL,ES:[FTERR]      ;CHECK FATAL MESSG FOR
042A 2402             512         AND   AL,02H             ; BAD CRC INDICATION
042C 750F             513         JNZ   SHORT BADRC        ;YES, JUMP
042E E4CC             514         IN    AL,SECRG           ;NO, GET SECTOR
0430 F6D0             515         NOT   AL                 ;###
0432 FEC0             516         INC   AL                 ; REGISTER AND
0434 F6D0             517         NOT   AL                 ;###
0436 E6CC             518         OUT   SECRG,AL           ; INCREMENT
0438 FECD             519         DEC   CH
043A 75E7             520         JNZ   VER                ;CONTINUE UNTIL B=0,
043C C3               521         RET                      ;THEN RETURN TO DISK
043D E4CC             522   BADRC:IN    AL,SECRG           ;LOAD REG B WITH
043F F6D0             523         NOT   AL                 ;###
0441 8AE8             524         MOV   CH,AL              ; BAD SECTOR NMBR
0443 C3               525         RET                      ;RETURN TO DISK
                      526   VERFY ENDP;
0444                  527   VRCRC PROC  NEAR
0444 BB2400           528         MOV   BX,IOPB4
0447 268A870000   R   529         MOV   AL,ES:[BX]
044C F6D0             530         NOT   AL
044E E6CC             531         OUT   SECRG,AL
0450 B180             532         MOV   CL,80H             ;SET LOOP COUNTER
0452 B004             533         MOV   AL,04H             ;LOAD READ INSTRUCTION
0454 F6D0             534         NOT   AL                 ;###
                      535                                  ; INTO COMMAND REGISTER
0456 B2C8             536         MOV   DL,STARG     ; DL=STATUS PORT
0458 B402             537         MOV   AH,02H       ; AH=MASK
045A BB3900           538         MOV   BX,BITBU      ;LOAD BIT BUCKET ADDRS TO BX
045D E6C8             539         OUT   COMRG,AL
045F                  540   LOOPV:                     ;
045F EC               541         IN    AL,DX        ;READ STATUS REGISTER
0460 22C4             542         AND   AL,AH        ; UNTIL DRQ
0462 75FB             543         JNZ   LOOPV        ; IS ACTIVE
0464 E4CE             544         IN    AL,DATRG     ; READ A BYTE TO BIT BUCKET
```

```
LOC  OBJ                LINE   SOURCE

0466 F6D0               545            NOT     AL                      ; READ AND DUMP DATA TO BIT BUCKET
0468 88870000       R   546            MOV     DS:MI[BX],AL            ;
046C EC                 547    DRQQ:   IN      AL,DX                   ; READ STATUS
046D 22C4               548            AND     AL,AH                   ; UNTIL DRQ
046F 74FB               549            JZ      DRQQ                    ; IS INACTIVE
0471 FEC9               550            DEC     CL                      ; DECREMENT COUNT
0473 75EA               551            JNZ     LOOPV
                        552                                    ;CHECK FOR DONE
0475 E4C8               553    ENDVR:  IN      AL,STARG        ; READ STATUS
0477 F6D0               554            NOT     AL              ; ###
0479 2401               555            AND     AL,01H          ; CHECK BUSY BIT
047B 75F8               556            JNZ     ENDVR           ; UNTIL ZERO
047D E4C8               557            IN      AL,STARG                ;GET STATUS WORD
047F F6D0               558            NOT     AL                      ;###
0481 2408               559            AND     AL,08H                  ;CHECK CRC BIT
0483 7501               560            JNZ     SHORT L_0
0485 C3                 561            RET
0486                    562    L_0:                                    ;RET IF GOOD
0486 B002               563            MOV     AL,02H                  ;SET ERROR MESSAGE
0488 26A22F00       R   564            MOV     ES:MLFTERR[],AL         ;IF BAD
048C C3                 565            RET
                        566    VRCRC   ENDP;
                        567
                        568                ;DELAY::: SOFTWARE TIME DELAY OF
                        569                ; [10(BX)+2] MICROSECONDS. 6.5 SEC>DELAY>12 MIC
048D                    570    DELAY   PROC    NEAR;
048D 90                 571            NOP             ;1.5 MIC
048E 90                 572            NOP             ;1.5 MIC
048F 90                 573            NOP             ;1.5 MIC
0490 4B                 574            DEC  BX         ;1.5 MIC
0491 75FA               575            JNZ  DELAY      ;4 MIC IF JUMP, 2 IF NOT
0493 C3                 576            RET             ;4 MIC
                        577    DELAY   ENDP;
                        578    ASMCODE ENDS
                        579            END
```

SYMBOL TABLE LISTING

NAME	TYPE	VALUE	ATTRIBUTES
??SEG	SEGMENT		SIZE=0000H PARA PUBLIC
ABS_0	SEGMENT		SIZE=0000H BYTE
ASMCODE	SEGMENT		SIZE=0494H BYTE
BADRC	L NEAR	043DH	ASMCODE
BCRC	NUMBER	002BH	
BITBU	NUMBER	0038H	
CGROUP	GROUP		ASMCODE CODE CONST DATA STACK MEMORY
CHKDO	L NEAR	02C5H	ASMCODE
CHKLD	L NEAR	029FH	ASMCODE
CHLD	L NEAR	0392H	ASMCODE
CHRST	NUMBER	00C4H	
CLR	L NEAR	00A6H	ASMCODE
CLRER	L NEAR	009FH	ASMCODE
CODE	SEGMENT		SIZE=0000H WORD PUBLIC 'CODE'
COMRG	NUMBER	00C8H	
CONC	L NEAR	011EH	ASMCODE
CONST	SEGMENT		SIZE=0000H WORD PUBLIC 'CONST'
CRCC	L NEAR	0241H	ASMCODE
CRE	L NEAR	0254H	ASMCODE
DATA	SEGMENT		SIZE=0000H WORD PUBLIC 'DATA'
DATRG	NUMBER	00CEH	
DELAY	L NEAR	048DH	ASMCODE
DERR	NUMBER	0035H	PUBLIC
DGROUP	GROUP		ASMCODE CODE CONST DATA STACK MEMORY
DISK	L FAR	0000H	ASMCODE PUBLIC
DR0TR	NUMBER	0031H	
DR1Q	L NEAR	0067H	ASMCODE
DR1TR	NUMBER	0032H	
DR2Q	L NEAR	0074H	ASMCODE
DR2TR	NUMBER	0033H	
DR3Q	L NEAR	0081H	ASMCODE
DR3TR	NUMBER	0034H	
DRNBR	NUMBER	0030H	
DRQAK	NUMBER	00C2H	
DRQQ	L NEAR	046CH	ASMCODE
DRSEL	L NEAR	0088H	ASMCODE
DRSLE	NUMBER	00C0H	
DRV1	L NEAR	0130H	ASMCODE
DRV2	L NEAR	013EH	ASMCODE
DRV3	L NEAR	014CH	ASMCODE
ENDRD	L NEAR	022DH	ASMCODE
ENDSK	L NEAR	01ABH	ASMCODE
ENDVR	L NEAR	0475H	ASMCODE
ENDWR	L NEAR	0344H	ASMCODE
ENTR1	L NEAR	0013H	ASMCODE
EXITSK	L NEAR	01BBH	ASMCODE
FATAL	L NEAR	01EDH	ASMCODE
FATE	L NEAR	0295H	ASMCODE
FCRC	L NEAR	03D3H	ASMCODE

```
FINSH . . . L NEAR   0154H  ASMCODE
FLAG1 . . . L NEAR   020AH  ASMCODE
FLAGW . . . L NEAR   0320H  ASMCODE
FTAL. . . . L NEAR   0388H  ASMCODE
FTERR . . . NUMBER   002FH   PUBLIC
IOPB. . . . V WORD   0000H  EXTRN
IOPB0 . . . NUMBER   0020H
IOPB1 . . . NUMBER   0021H
IOPB2 . . . NUMBER   0022H
IOPB3 . . . NUMBER   0023H
IOPB4 . . . NUMBER   0024H
IOPB5 . . . NUMBER   0025H
IOPB6 . . . NUMBER   0026H
IOPB7 . . . NUMBER   0027H
IOPB8 . . . NUMBER   0028H
IOPB9 . . . NUMBER   0029H
IOPBC . . . NUMBER   0036H
L_5 . . . . L NEAR   01BCH  ASMCODE
L_6 . . . . L NEAR   02E6H  ASMCODE
L_7 . . . . L NEAR   03ECH  ASMCODE
L_8 . . . . L NEAR   0486H  ASMCODE
L01 . . . . L NEAR   0021H  ASMCODE
LDLT. . . . L NEAR   02BDH  ASMCODE
LOOPV . . . L NEAR   045FH  ASMCODE
 OSTD . . . NUMBER   002DH
 STD. . . . L NEAR   03ACH  ASMCODE
M . . . . . V BYTE   0000H  ABS_0
MEMORY. . . SEGMENT         SIZE=0000H WORD  MEMORY 'MEMORY'
MEMORY_ . . V BYTE   0000H  MEMORY
NOWR. . . . L NEAR   0107H  ASMCODE
NXTR. . . . L NEAR   03DDH  ASMCODE
OP3 . . . . L NEAR   00C3H  ASMCODE
OP4 . . . . L NEAR   00D8H  ASMCODE
OP5 . . . . L NEAR   00E4H  ASMCODE
OP6 . . . . L NEAR   00EDH  ASMCODE
OP7 . . . . L NEAR   00FEH  ASMCODE
OPER. . . . L NEAR   00B4H  ASMCODE
PRNF. . . . NUMBER   002EH
PRST. . . . NUMBER   002CH
R1. . . . . L NEAR   00CDH  ASMCODE
R2. . . . . L NEAR   00D3H  ASMCODE
READ. . . . L NEAR   01F4H  ASMCODE
RECNF . . . L NEAR   0358H  ASMCODE
REP_. . . . L NEAR   02D7H  ASMCODE
REPT. . . . L NEAR   015DH  ASMCODE
REST. . . . L NEAR   01D9H  ASMCODE
RESTR . . . L NEAR   0277H  ASMCODE
RLOOP . . . L NEAR   021BH  ASMCODE
RNF . . . . L NEAR   025BH  ASMCODE
RSET. . . . L NEAR   036BH  ASMCODE
 ECRG . . . NUMBER   00CCH
 EEK. . . . L NEAR   016BH  ASMCODE
SELDR . . . L NEAR   0050H  ASMCODE
SKIP. . . . L NEAR   0116H  ASMCODE
STACK . . . SEGMENT         SIZE=0000H WORD  STACK 'STACK'
```

```
STACK_BASE  V BYTE   0000H  STACK
STARG . . . NUMBER   00C8H
STPB. . . . L NEAR   0026H  ASMCODE
SWSV. . . . NUMBER   002AH
THRU. . . . L NEAR   010EH  ASMCODE
TLOOP . . . L NEAR   0331H  ASMCODE
TRKRG . . . NUMBER   00CAH
VER . . . . L NEAR   0423H  ASMCODE
VERFY . . . L NEAR   0410H  ASMCODE
VERIF . . . L NEAR   03B4H  ASMCODE
VRCRC . . . L NEAR   0444H  ASMCODE
WLOOP . . . L NEAR   0335H  ASMCODE
WR. . . . . L NEAR   00F1H  ASMCODE
WRITE . . . L NEAR   030AH  ASMCODE


ASSEMBLY COMPLETE, NO ERRORS FOUND
```

IV. LINKING AND LOCATING THE USAFA/8086 SOFTWARE MODULES

The information presented in this chapter is essential to the user's understanding of how to locate the variables and code in the various USAFA/8086 modules. The linkage map conveys little information to the user other than all external references were satisfied. The locate map, however, presents the segment and offset of all public procedures, labels, and variables encountered in the USAFA/8086 modules. Additionally, the start address of all modules, data segments, and code segments as well as the system stack are given. The stack location given (00000H) must be offset by 200H to determine the actual stack position as this is altered by the PLM86 code of USAF86. The location of any variable can be determined using the information given in the locate map in conjunction with the cross reference map produced by the computer or assembler. For example, suppose we desire to find the variable CHAR in USAF86. From the locate map USAF86.DATA begins at 00250H. From the cross reference map of the compilation of USAF86, the variable CHAR is located relatively at 004C . By hexadecimal addition 00250+004C=0029CH. Since CHAR happens to be PUBLIC we will also see a direct location in the locate map. It is shown in segment 0025H at offset 004CH. This also implies the absolute address of 0029CH. The listings of the linkage and locate maps follow.

```
ISIS-II MCS-86 LINKER V4.1 INVOKED BY:
LINK86 :F1:CURRNT.OBJ, :F1:FITZ.OBJ, :F1:DISK.OBJ TO :F1:CURRNT.LNK &
MAP SB PL PR(:F1:LINK.LST)
LINK MAP FOR :F1:CURRNT.LNK(USAF86)

LOGICAL SEGMENTS INCLUDED
LENGTH ADDRESS  SEGMENT         CLASS
 2236H ------   USAF86.CODE     CODE
 0074H ------   USAF86.DATA     DATA
 00ACH ------   STACK           STACK
 0000H ------   MEMORY          MEMORY
 2722H ------   DOS.CODE        CODE
 00A5H ------   DOS.DATA        DATA
 0000H ------   ??SEG
 0494H ------   ASMCODE
 0000H ------   CODE            CODE
 0500H ------   CONST           CONST
 0000H ------   DATA            DATA
 0000H ------   ABS_0

INPUT MODULES INCLUDED:
:F1:CURRNT.OBJ(USAF86)
:F1:FITZ.OBJ(DOS)
:F1:DISK.OBJ(DISK)
```

```
ISIS-II MCS-86 LOCATER, V1.1 INVOKED BY:
LOC86 :F1:CURRNT.LNK TO :F1:CURRNT MA SB PL PR(:F1:LOC.LST) &
BS ST(MAINPROGRAM) OD(SM(STACK,USAF86.DATA,DOS.DATA,USAF86.CODE, &
ASMCODE)) &
AD(SM(STACK(00000H),USAF86.DATA(00250H),USAF86.CODE(0FB000H)))

SYMBOL TABLE OF MODULE USAF86
READ FROM FILE :F1:CURRNT.LNK
WRITTEN TO FILE :F1:CURRNT

BASE   OFFSET TYPE SYMBOL              BASE   OFFSET TYPE SYMBOL

0025H  0000H  PUB  MEMORYARG1PTR       0025H  004CH  PUB  CHAR
0003H  0001H  PUB  DISKTRACK           0003H  0006H  PUB  IOPB
FB00H  135CH  PUB  TIMERREAD           FB00H  138DH  PUB  TIMERLOAD
FB00H  12D1H  PUB  WRITECONSOLELEDS    FB00H  12C3H  PUB  READCONSOLESWITC
                                                         -HES
FB00H  0F5FH  PUB  CONVERTTOFLOAT      FB00H  0F51H  PUB  ALUSTATUSWORD
FB00H  0F3EH  PUB  ALUCMNDWORD         FB00H  0F10H  PUB  ALUINWORD
FB00H  0EE5H  PUB  ALUOUTWORD          FB00H  0EB3H  PUB  USERIOGETCHAR
FB00H  0E98H  PUB  SUPPRESSBLANKS      FB00H  0DDFH  PUB  DISPLAYDATAARRAY
FB00H  0D8CH  PUB  SIONEWLINE          FB00H  0D7BH  PUB  SIOCRLF
FB00H  0D51H  PUB  SIOBLANKS           FB00H  0D24H  PUB  SIOGETADDR
FB00H  0C5AH  PUB  SIOGETWORD          FB00H  0B0CH  PUB  SIOGETBYTE
FB00H  0B89H  PUB  SIOHEX              FB00H  0B6CH  PUB  SIOVALIDHEX
FB00H  0B30H  PUB  SIOGETCHAR          FB00H  0AF9H  PUB  SIOOUTSTRING
FB00H  0AD7H  PUB  SIOOUTWORD          FB00H  0AA3H  PUB  SIOOUTBYTE
FB00H  0A77H  PUB  SIOOUTCHAR          FB00H  0A4AH  PUB  FORCEUPPERCASE
FB00H  0A31H  PUB  KEYBDCHARIN         FB00H  0A18H  PUB  SERIALCHARIN
FB00H  097DH  PUB  CRTOUT              FB00H  0964H  PUB  SERIALCHAROUT
FB00H  08C2H  PUB  DELAYLONG           FB00H  08A8H  PUB  DELAY
FB00H  088FH  PUB  LPNEWLINE           FB00H  087BH  PUB  LPCRLF
FB00H  0851H  PUB  LPBLANKS            FB00H  081AH  PUB  LPOUTSTRING
FB00H  07F8H  PUB  LPOUTWORD           FB00H  07C4H  PUB  LPOUTBYTE
FB00H  079EH  PUB  LPOUTCHAR           FB00H  06B4H  PUB  NEXTCOMMAND
FB00H  0764H  PUB  ERROR               FB00H  04E1H  PUB  MAINPROGRAM
FD6CH  237EH  PUB  DIRECTORYLIST       FD6CH  2220H  PUB  INITIALIZE
FD6CH  1D2AH  PUB  ATTRIBUTECHANGED    FD6CH  1CD4H  PUB  ERASEDRIVER
                   -RIVER
FD6CH  1C3EH  PUB  RENAMEDRIVER        FD6CH  1B9AH  PUB  COPYDRIVER
FD6CH  1B4DH  PUB  WRITEDRIVER         FD6CH  1AFBH  PUB  READDRIVER
0000H  002FH  PUB  FTERR               0000H  0035H  PUB  DERR
FD23H  0006H  PUB  DISK
```

```
MEMORY MAP OF MODULE USAF86
READ FROM FILE  F1 CURRNT LNK
WRITTEN TO FILE :F1 CURRNT

MODULE START ADDRESS  PARAGRAPH = FB00H  OFFSET = 04E1H
SEGMENT MAP

START     STOP     LENGTH ALIGN NAME            CLASS

00000H   00ABH     00ACH   W   STACK           STACK
00250H   02C3H     0074H   W   USAF86_DATA     DATA
02C4H    0368H     00A5H   W   DOS_DATA        DATA
FB000H   FD235H    2236H   W   USAF86_CODE     CODE
FD236H   FD6C9H    0494H   B   ASMCODE
FD6CAH   FFDEBH    2722H   W   DOS_CODE        CODE
FFDECH   FFDECH    0000H   W   CODE            CODE
FFDF0H   FFDF0H    0000H   G   ??SEG
FFDF0H   FFDF0H    0000H   B   ABS_0
FFDF0H   FFDF0H    0000H   W   CONST           CONST
FFDF0H   FFDF0H    0000H   W   DATA            DATA
FFFF0H   FFFF4H    0005H   A   (ABSOLUTE)
FFFF6H   FFFF6H    0000H   W   MEMORY          MEMORY


GROUP MAP

ADDRESS  GROUP OR SEGMENT NAME
FD230H   DGROUP
         STACK
         MEMORY
         ASMCODE
         CODE
         CONST
         DATA
FD230H   CGROUP
         STACK
         MEMORY
         ASMCODE
         CODE
         CONST
         DATA
```

V. CLOSING COMMENTS

During the development of the USAFA/8086 software, several problems were encountered with the Intel MDS-311 Software Package which remain unresolved as of this time.

1. The bootstrap function of the locator does not work properly.

2. The ASM86 assembler insists on occupying all 64K of the MDS800 disallowing the standard interface to user files such as :L1: (my line printer!).

3. The PLM86 compiler has at least one error in it. Whenever any comparison is made to the literal 0 (zero) the compiler generates erroneous code. Example: IF A=0 THEN .... will generate code which checks the value of the ZF generated by the last ALU operation rather than the value of A itself.

4. LINK86 balks at certain results given by ASM86.

5. The compiler name table was repeatedly exhausted during the development of DOS86.

The source software produced under government research is available for a limited time by the requestor supplying a single sided single density diskette to:

USAFA/DFEE/Fairchild Hall
Computer Systems Laboratory Room 2H25
USAF Academy, Colorado 80840

APPENDIX A

USAFA/8086 OPERATOR'S GUIDE

The following steps are required for system initialization.

STEP	RESULT
1 Machine to Single Step	-
2 AC Power to on (DOWN)	Panel lights on
3 Push Reset	Display Adrs: FFFF0 Data: E1EA
else push again	
4 Machine to run	Display FbE68 8888 (Running)
else repeat from 3	
5 Identify console by space bar depress	Logo to console Display Fb888 8888 (Running) Logo:

```
USAFA 8086 VER 3.20
DISK 0 NOT READY
DISK 1 NOT READY
ENTER COMMAND PLEASE:
```

Operating System Logo at Initialization.

6 System Ready for Operation

The following presents a brief summary of the USAFA/8086.

Console Description

Power Switch: UP-OFF DOWN-ON
16 LED display: Output from Port FE/FF
16 Switches: Input to Port FE/FF
Switches and Leds are not electrically connected.

Single Step Unit

Displays current address and contents. Address is shown in absolute (physical) form between 00000_H and $FFFFF_H$. Data may assume any value between 0000_H and $FFFF_H$. It represents instruction, data, or address information depending on relative byte position. To the right are 4 discrete leds representing the major control lines of the system:

MEMR - Memory Read
IOR - Input/Output Read
IOW - Input/Output Write
MEMW - Memory Write

At system initialization, the unit may be left in single step mode. Below the displays is a push button allowing single stepping. Following a system reset we find the following:

AT SYSTEM RESET	ADDRS	DATA	REMARKS
	FFFF0	E1EA	FFFF0 is the 8086 restart address. EA is an extended jump op code. E1 is the offset low byte.
Push Step	FFFF2	0004	04 is the offset high byte total offset=$04E1_H$. 00 is the segment low byte.
Push Step	FFF4	00Fb	Fb is the high byte of the segment. Total segment=$Fb00_H$. Absolute address = Fb00 + 04E1 Fb4E1 This is the start of the operating system determined at the time of location using LOC86 by the software designer.
Push Step	FFFF6	29C7	This step is the bus interface unit fetching ahead to the instruction queue. Actual data is not meaningful since it will not be executed as soon as the processor realizes a JMP has been ordered.
Push Step	Fb4E1	bCFb	These are the first two bytes of the actual operating system (USAF86 Module).

Reset Button

The reset button asserts a signal to the microprocessor forcing it to resume processing at location FFFF0. This cannot be changed as it is an internal function of the CPU NMI. This button forces an interrupt which the processor cannot ignore. The operating system loads locations 8, 9, A, and B with EF, 1F, 00, FB. This is the address of the operating system routine service routine for the NMI. The vector location 8 to B cannot be changed. It's contents, however, could be reloaded to any value from software.

The following section describes the primary control characters and gives examples of typical system reaction. The operating system module USAF86 is responsive to the following primary control characters:

S - Substitute
L - List
G - Go
F - Fill
M - Move
P - Print
Q - Convert
B - Breakpoint Set
C - Clear Breakpoint
T - Test Module
X - Abort

The operating system module DOS86 uses the following primary control characters:

R - Read
W - Write
D - Directory
I - Input Hex Files
N - Name Change
A - Attribute change
K - Copy

Any primary character evokes a full English response with prompts from USAF86 or DOS86.

Substitute (S): Allows the user to alter memory byte by byte. The system asks for an address. This may be in the form XXXX:YYYY±DDDD where XXXX is the desired segment. YYYY is the desired offset. DDDD is the desired displacement. Entry is terminated by a Carriage Return. The system responds by giving a display of the net offset followed by the current memory value. Cells which are not desired to be changed can be passed using either the space bar or the carriage return. Routine is exitted using the character X which is the standard system abort character. 0's are understood by default if not entered, i.e., 8=0008.

Substitute Command Examples

```
X
ENTER COMMAND PLEASE:SUBSTITUTE (ADRS)
•FFFF:0000
0000-FA-X
ENTER COMMAND PLEASE:SUBSTITUTE (ADRS)
•0000:100X
ENTER COMMAND PLEASE:SUBSTITUTE (ADRS)
•0000:0100
0100-2F-11
0101-00-22
0102-F0-33
0103-F7-44
0104-0F-55
0105-80-X
ENTER COMMAND PLEASE:SUBSTITUTE (ADRS)
•100
0100-11-12
0101-22-23
0102-33-34
0103-44-45
0104-55-56
0105-80-X
ENTER COMMAND PLEASE:SUBSTITUTE (ADRS)
•100+6
0106-F0-78
0107-F7-89
0108-0F-9A
0109-00-X
```

Aborted Sequences

Segment: Offset

Offset Only, Leading Zeroes Not Shown.

Offset + Displacement (Segment Implied)

List (L): Allow the user to display to the console any contiguous block of memory. User must supply the address in format (XXXX:YYYY) and the number of bytes to be displayed. Entries are in hex.

```
ENTER COMMAND PLEASE:LIST (ADRS,CNT)
*0105,25
0105                  80 78 89 9A 00 F0 FB 0F 00 F0 F7
0110   8F 58 F0 F8 0F 00 F0 FF 0F C0 F0 FE 0F 82 F0 FC
0120   0B 00 F0 FD 0F 04 F0 FF 0E 45
```

List Command Example

Go (G): Allows the user to execute a program at a given address in format XXXX:YYYY. Again, leading zeros and unentered data are assumed as zeroes.

```
ENTER COMMAND PLEASE:GO (ADRS)
*FB00:06R4

ENTER COMMAND PLEASE:
```

Execution Example
Execute "Next Command"

Fill (F): Allows the user to block fill any contiguous block in memory. User must supply the address, the number of bytes, and value to be filled in byte form (00-FF).

```
ENTER COMMAND PLEASE:FILL (START,CNT,VAL)
*2000,100,00
ENTER COMMAND PLEASE:LIST (ADRS,CNT)
*2000,20
2000   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
2010   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Fill Example and
Partial Result

Move (M): Allows the user to move any block to any location. Address of source, address of destination, and number of bytes must be supplied. Data moved is copied exactly.

```
ENTER COMMAND PLEASE:MOVE (SOURCE,DEST,CNT)
*100,2000,10
ENTER COMMAND PLEASE:LIST (ADRS,CNT)
*2000,20
2000   12 23 34 45 56 80 78 89 9A 00 F0 FB 0F 00 F0 F7
2010   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Move Example: Data at 100 Entered Above; See Substitute Example

Print (P): Allows contiguous block of memory to be dumped to the line printer. User supplies address and count.

```
0000   0F 50 F0 BF F0 00 0F AF 0F 0E F0 77 F0 8C 0A 5F
00E0   0F 40 F0 FE F0 A9 0F 5F 0F D0 F0 FE F0 00 0F BE
00F0   8F 10 F0 FF F0 1A 0F EF 0F CC F0 FF F0 A8 0F 5F
```

```
8086 MEMORY DUMP

2000   12 23 34 45 56 80 78 89 9A 00 F0 FB 0F 00 F0 F7
2010   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Convert (Q): Changes a given representation of a floating point word to its decimal value and prints it on the console. User supplies the address of the floating point number. Routine uses AM9511 APU.

```
ENTER COMMAND PLEASE:SUBSTITUTE (ADRS)
•3000
3000-00-02
3001-BF-80
3002-FD-00
3003-20-00
3004-00-X
```

Substitute a 2.0 at location 3000

```
ENTER COMMAND PLEASE:Q 3000
     +2.00000000 E-00
```

Q (Convert) Example

Breakpoints (B): Can be set any address in RAM. User supplies the address. Will restore call upon encountering breakpoint. Uses software interrupt 3. Only one breakpoint is allowed.

Clear Breakpoint (C): Allows a user to clear a unencountered breakpoint. Caution: clearing nonexistant breakpoints has an undefined result.

Breakpoint Operations

```
ENTER COMMAND PLEASE:BREAKPOINT SET (ADRS)
•1000
ENTER COMMAND PLEASE:GO (ADRS)
•1000
CLEAR BREAKPOINT

BREAK POINT ENCOUNTERED AT 0000:1000
```

Set Breakpoint at 1000

Execute 1000

Breakpoint Encountered and Cleared

Breakpoint Found

```
ENTER COMMAND PLEASE:LIST (ADRS,CNT)
•1000,1
1000   0C  ← Contents of 1000 Before Breakpoint
ENTER COMMAND PLEASE:BREAKPOINT SET (ADRS)
•1000
ENTER COMMAND PLEASE:LIST (ADRS,CNT)
•1000,1
1000   CD  ← Interrupt 3 (Software) set by operating system
ENTER COMMAND PLEASE:CLEAR BREAKPOINT

ENTER COMMAND PLEASE:LIST (ADRS,CNT)
•1000,1
1000   0C  ← Original Content Restored
```

Test Module (T): Invokes system diagnostic package. Secondary command characters used are:

- M - Memory test
- A - APU test
- R - Ramp DAC test
- H - Hybrid Loop test
- L - Loop tolerance test
- C - Console LEDs/Switches
- I - Integrated Circuitry
- K - Keyboard Display
- T - Timer
- D - Discrete
- E - Console LED Ramp
- F - Floppy Disk

Memory Test (T,M): user supplies starting address. Memory is written twice; first to 55 and then to AA one cell at a time. First cell to fail on readback is declared bad. Hence, routine must fail at the top of memory. Starting address should be greater than 00369H since system data and stack are below this level.

Memory Test Example

```
ENTER COMMAND PLEASE:TEST MODULE
*MEMORY TEST
ENTER STARTING ADDRESS:369
MEMORY FAILED AT 0000:8000 WITH THE VALUE FF
ENTER COMMAND PLEASE:LIST (ADRS,CNT)
*1000,20
1000   AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA
1010   AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA
```

← Must fail at top of memory.

Result of all memory being tested!

APU Test (T,A): Works a "simple" problem in floating point using the AM9511. The test actually is the conversion which is presented to the console. Uses the convert function of USAF86. Should show:

```
ENTER COMMAND PLEASE:TEST MODULE                APU Test
*APU TEST

     +2.0000000 E-00 +       +3.0000000 E-00=      +5.0000000 E-00
```

Ramp Test (T,R): Ramps the indicated DAC (ports 40 to 5E) from -100 volts to +100 volts. An o'scope should be used to analyze channel performance. Exit by NMI.

```
ENTER COMMAND PLEASE:TEST MODULE      Hybrid Ramp Test Examples
*RAMP DACS
ENTER PORT NUMBER:42
```

Hybrid Loop test (T,H): Takes a given ADC port (40-5E) and an arbitrary DAC port (40-5E) and links them through software connecting the DAC port to the ADC port. Exit by NMI.

```
ENTER COMMAND PLEASE:TEST MODULE,    Hybrid Loop Test Example
*HYBRID LOOP
ENTER DAC PORT:42                    Result:  DAC 42 tied to ADC 50
ENTER ADC PORT:50
```

Hybrid Tolerance Test: allows a tolerance check of ADC/DAC loop. Not used thus far with system.

```
ENTER COMMAND PLEASE:TEST MODULE                Hybrid Loop Tolerance Test
*LOOP TEST
ENTER DAC VALUE: 42
ENTER PORT NUMBER: 44
HYBRID LOOP ERROR WAS GREATER THAN TOLERANCE--015D
ENTER DAC VALUE: X
```

Console/LED Switches (TC): Console switches are software connected to console leds (i.e., they echo). Exit by NMI.

```
ENTER COMMAND PLEASE:TEST MODULE          Console/LED Loop Test
*CONSOLE LEDS/SWITCHES
```

Integrated Circuit Test (T,I): This routine allows the user to directly address any 8 bit input or output port in the system. If a read is desired the result is given the console leds. If a write is directed, the data is taken from the console switches. X to abort.

```
ENTER COMMAND PLEASE:TEST MODULE                I/O Tests
*INTEGRATED CIRCUITRY
ENTER PORT NUMBER FF
TYPE R TO READ, W TO WRITE R
ENTER PORT NUMBER FE
TYPE R TO READ, W TO WRITE R
ENTER PORT NUMBER FF
TYPE R TO READ, W TO WRITE R
ENTER PORT NUMBER FE
TYPE R TO READ, W TO WRITE W
ENTER PORT NUMBER FF
TYPE R TO READ, W TO WRITE W
```

Keyboard/display Test (T,K): Has the user print in the entire alphabet and numerics.

```
ENTER COMMAND PLEASE:TEST MODULE                    Keyboard Test
*KEYBOARD DISPLAY TEST
ENTER THE EXACT SAME DATA
ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789
ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789
ENTER COMMAND PLEASE:TEST MODULE
*KEYBOARD DISPLAY TEST
ENTER THE EXACT SAME DATA
ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789
ABCDF
KEYBD ENTRY ERROR
```

Timer Test (T,I): Causes the 8253 to be programmed so that it outputs 50% duty cycle square waves.

OUT0 = 1000 Hz
OUT1 = 1000 Hz
OUT2 = 1000 Hz

Square waves must be checked with an o'scope at the 8253 on the hybrid interface card. timer continues running in this mode until reprogrammed.

```
ENTER COMMAND PLEASE:TEST MODULE          Timer Test Example
*TIMER TEST
```

Discrete Test (TD): The discrete out and in ports must be cabled to each other and 28 volt power applied. Each port has all values from 0000 to FFFF written and verified.

```
ENTER COMMAND PLEASE:TEST MODULE          Discrete I/O
*DISCRETE I/O TEST

DISCRETE TEST 1 FAILED--0001          Unit Not Cabled Properly
```

Test Console Leds (TE): Ramps the console leds (i.e., this is a busy work routine - good for those who like flashing lights!). Exit by NMI.

```
ENTER COMMAND PLEASE:TEST MODULE      Console LED Ramp Test
*CONSOLE LED RAMP
```

Test Floppy Disk: Assumes disk powered and diskette inserted and ready. Allows the user to directly access any track and sector of either diskette. No software write protects are honored. User must provide the IOPB-Input Output Parameter Block in memory.

IOPB

WORD	
0	Channel Word, normally=00
1	Diskette Word, indicate drive, operation 04=Read,0 14=Read,1 06=Write,0 16=Write,1 01=Seek,0 11=Seek,1
2	# of sectors to be affected, usually=01
3	Track (0-4A)
4	Sector (0-1A)
5	Offset Buffer Addrs Low } Assumed in Segment 0000
6	Offset Buffer Addrs High } Assumed in Segment 0000
7	IOPB # normally=01
8	Next IOPB Low, normally=00
9	Next IOPB High, normally=00

```
ENTER COMMAND PLEASE:SUBSTITUTE (ADRS)      Floppy Disk Test
•1000                                       Create IOPB
1000-AA-00  — Channel Word
1001-AA-03  — Diskette Word
1002-AA-01  — # of Records
1003-AA-00  — Track
1004-AA-01  — Sector
1005-AA-00  — Buffer Low Offset
1006-AA-07  — Buffer High Offset
1007-AA-01  — IOPB #
1008-AA-00  — Next IOPB Low Offset
1009-AA-00  — Next IOPB High Offset
100A-AA-X
```

The disk operating system uses memory locations up to 0780H. Each sector of the diskette contains 80_H or 128_{10} bytes in IBM3740 format. The following primary control characters invoke disk operating system routines:

D - Directory
N - Name change
A - Attribute Change
R - Read
W - Write
I - Input Hex File
K - Copy
E - Erase

Directory (D): The user must indicate drive number (0 or 1) and whether (A) all files are desired.

```
ENTER COMMAND PLEASE:DIRECTORY 0,A
FILE     EXT   DATASECTORS   ATTR

ISIS     DIR       0019      FI
ISIS     MAP       0002      FI
ISIS     TO        0017      FI
ISIS     LAB       0001      FI
ISIS     BIN       005C      SFI
ISIS     CLI       0013      SFI
ASM80              006C      SWI
ASM80    OV0       0010      SWI
ASM80    OV1       0010      SWI
ASM80    OV2       0011      SWI
ASM80    OV3       0008      SWI
ASM80    OV4       0089      SWI
ASXREF             0022      SWI
FORMAT             003D      SWI
COPY               0042      SWI
DELETE             0026      SWI
DIR                0036      SWI
EDIT               0039      SWI
HEXOBJ             0021      SWI
OBJHEX             001B      SWI
RENAME             0013      SWI
SWITCH             0006      SWI
TD822              001E      SWI
JOE      TST       0006
DEMO     JOE       0002
TOTAL  SECTORS= 03C9/07D2
```

Directory Command Example

All Files

```
ENTER COMMAND PLEASE:DIRECTORY 0
FILE     EXT   DATASECTORS

JOE      TST       0006
DEMO     JOE       0002
TOTAL  SECTORS= 03C9/07D2
```

Directory Example

NON-I Files

Name Change (N): The user must supply the drive (format :F0: or :F1:) and the file name.

```
ENTER COMMAND PLEASE:NAMECHANGE                    Name Change
*:F0:JOE.TST,JIM.TST
ENTER COMMAND PLEASE:DIRECTORY 0
FILE     EXT   DATASECTORS

JIM      TST       0006
DEMO     JOE       0002
TOTAL SECTORS= 03C9/07D2
```

Attribute Change: The user must supply the drive number, the file name, and the attribute typing as:

NI	**Not Invisible**
I	**Invisible**
NW	**Not Write Protected**
W	**Write Protect**
NF	**Not Format**
F	**Format**
NS	**Not System**
S	**System**

```
ENTER COMMAND PLEASE:ATTRIBUTE CHANGE          Attribute Change
*:F0:JIM.TST,I
ENTER COMMAND PLEASE:DIRECTORY 0               Make I
FILE     EXT   DATASECTORS

DEMO     JOE       0002
TOTAL SECTORS= 03C9/07D2
```

```
ENTER COMMAND PLEASE:ATTRIBUTE CHANGE
*:F0:JIM.TST,NI
ENTER COMMAND PLEASE:DIRECTORY 0               Return to NON-I
FILE     EXT   DATASECTORS

JIM      TST       0006
DEMO     JOE       0002
TOTAL SECTORS= 03C9/07D2
```

Read (R): Read copies identically the contents of a file into memory. User provides drive, name, start address, and upper limit. The start address will default to 1000H and the upper limit will default to 07FFFH.

```
ENTER COMMAND PLEASE:READ
*:F0:DEMO.JOE
READ OPERATION COMPLETED
```

Read Example

Input Hex File (I): Reads and interprets a hex file placing it in the location in memory indicated within the file. See the Intel 8086 manual for absolute file formats. This is the method by which we download from the MDS800 to the USAFA 8086.

```
ENTER COMMAND PLEASE:INPUT HEX FILE
*:F0:DEMO.JOE
```

Input Hex File Example

Write (W): This routine copies from memory to disk exactly. User gives a drive, file name, start address, and final address.

```
ENTER COMMAND PLEASE:WRITE
*:F0:ECHO.BIN,1000,107F
WRITE OPERATION COMPLETED
```

Write Example

```
ENTER COMMAND PLEASE:DIRECTORY 0
FILE     EXT   DATASECTORS

JIM      TST       0006
DEMO     .DE       0002
ECHO     BIN       0001
TOTAL SECTORS= 03CB/07D2
```

New Directory

Kopy (K): Copies a file from drive to drive. User gives drive and name to drive and name.

```
ENTER COMMAND PLEASE:KOPY
*:F0:ECHO.BIN,:F1:ECHO.BUN
ENTER COMMAND PLEASE:DIRECTORY 1
FILE     EXT   DATASECTORS

DEMO     BIC       0002
ECHO     BUN       0001
TOTAL SECTORS= 03C4/07D2
```

Copy Example

New Directory of Drive 1

Erase (E): Delete a given file from a given drive.

Erase Example

```
ENTER COMMAND PLEASE:ERASE
*:F1:ECHO.BUN
ENTER COMMAND PLEASE:DIRECTORY 1
FILE     EXT   DATASECTORS

DEMO     RIC       0002
TOTAL SECTORS= 0303/07D2

ENTER COMMAND PLEASE:ERASE
*:F1:FAKE.BIN
FILE NOT FOUND
ENTER COMMAND PLEASE:
USAFA 8086 VER 3.20
DISK 0 NOT READY
DISK 1 NOT READY
ENTER COMMAND PLEASE:DIRECTORY
FILE     EXT   DATASECTORS


USAFA 8086 VER 3.20
DISK 0 READY
DISK 1 NOT READY
ENTER COMMAND PLEASE:DIRECTORY
FILE     EXT   DATASECTORS

BOOT  ~        C330
JIM      TST       0006
DEMO     JOF       0002
ECHO     BIN       0001
TOTAL SECTORS= 030B/07D2

ENTER COMMAND PLEASE:DIRECTORY
FILE     EXT   DATASECTORS

JIM      TST       0006
DEMO     JOF       0002
ECHO     BIN       0001
TOTAL SECTORS= 030B/07D2
```

Additional Erase Example

```
ENTER COMMAND PLEASE:ERASE
*:F0:ASMAO
FILE WRITE PROTECTED
```

APPENDIX B

MDS-800 DISKETTE PHYSICAL AND LOGICAL ORGANIZATION

MDS-800 DISKETTE PHYSICAL AND LOGICAL ORGANIZATION

Each diskette is physically organized as 77 recording tracks of 26 sectors (often called records or blocks) per track and 128 data bytes per sector. The actual recording of digital data is in accordance with the IBM 3740 standard format. For specific details on this format refer to the IBM Compatibility Reference Manual published by Shugart Associates.

Based upon this physical organization, the MDS-800 system software imposes an additional logical organization for recorded data. MDS-800 reserves specific tracks and sectors for system use and many of these must be preserved across MDS-800 and non MDS-800 systems for compatibility. These reserved tracks and sectors and their uses and formats are as follows:

TRACK Ø

a. System Initialization Program - Sectors 1 through 23 are used by MDS-800 for its System Initialization Program. Even though non-MDS systems may not require such a program (e.g., it may be in ROM instead) for compatibility with MDS systems these sectors should not be used for any other purposes.

b. Disk Label - Sector 26 is used to record the logical name assigned to the diskette by the user when prompted by MDS-800. Only the first 9 bytes of this sector are used to store the ASCII name which is assigned in conformance with the MDS convention of names composed of 1 to 6 characters with optional extensions of 0 to 3 characters (i.e., NNNNNN.EEE). For names shorter than 6 characters and/or extensions shorter than 3 characters, ASCII nuls ($ØØ_H$) are inserted so that 9 characters are always recorded.

EXAMPLES:

TRACK Ø, SECTOR 26

DISKETTE NAME	BYTE: 1	2	3	4	5	6	7	8	9
DELETE.[1]HEX ⟶	44[2]	45	4C	45	54	45	48	45	58
SAM ⟶	53	41	4D	ØØ	ØØ	ØØ	ØØ	ØØ	ØØ
ABLE.OBJ ⟶	41	42	4C	45	ØØ	ØØ	4F	42	4A
COPY.TO ⟶	43	4F	5Ø	59	ØØ	ØØ	54	4F	ØØ
	NAME						EXTENSION		

NOTES: (1) The ASCII character for the period (.) is not recorded.
(2) Hex representations of ASCII characters.

c. Linkage Maps - Sectors 24 and 25 are used by MDS-800 as linkage maps for the System Initialization Program and Disk Label respectively. Linkage maps are closely tied to the diskette directory so their purpose and format will be discussed after the directory is described.

TRACK 1

a. Directory - Sectors 2 through 26 are used by MDS-800 to store file directory information for all files stored on the disk. Each directory entry requires 16 bytes thus allowing up to 200 files per disk (assuming, of course, that the files are sufficiently small to all fit on the remaining tracks of the disk).
The format for each directory entry is:

BYTE	CONTENT/USE
1	00 = ACTIVE FILE: Active files are protected from being overwritten by newly created files and are readily accessable via all MDS commands. FF = INACTIVE FILE: The directory entry, sector linkages and actual file contents may still be intact for a file made inactive but since inactive file space is eligible for being overwritten by newly created files, re-activation of inactive files may only be possible if no other files have been created subsequent to file deactivation. 7F = UNUSED FILE ENTRY: Space not yet used for directory entries. MDS inserts nuls (00) in the remaining 15 bytes of all such unused file entries.
2-7	FILE NAME = 1 to 6 ASCII characters with nuls (00) in byte positions where less than 6 characters are used.
8-10	FILE NAME = 0 to 3 ASCII characters with nuls (00) EXTENSION in byte position where less than 3 characters are used.
11	FILE ATTRIBUTE FLAGS BIT 7 = FORMAT FLAG (F) - When set, the file cannot be deleted, renamed, or opened for output (i.e. write-to-file) or update. (Newly formed diskettes are also assigned this attribute.) This attribute is normally only given to files which should never be changed but not to any other files since they can be protected by the WRITE-PROTECT attribute flag. BITS 6-3 = Not used - set to 0 by MDS-800. BIT 2 = WRITE-PROTECT FLAG (W) - When set the file cannot be deleted, renamed, or opened for output or update.

BYTE	CONTENT/USE
	BIT 1 = SYSTEM FLAG (S) - Used in conjunction with the S switch in the MDS-800 FORMAT command to cause the file to be copied to the second disk drive (i.e., drive 1).
	BIT Ø = INVISIBLE FLAG (I) - Used in conjunction with the I switch in the MDS-800 DIR command to suppress listing of the file's directory entry.
12	FINAL SECTOR BYTE COUNT = Binary Count of the number of valid bytes in the last sector of the file. Valid bytes are those that actually belong in the file content versus filler bytes often inserted to fill out each sector. $Ø1 \rightarrow 8Ø_H$ are the only values this byte can assume.
13/14	# OF SECTORS IN THE FILE = 16 bit binary count of the number of sectors in the file. Byte 13 holds the least significant bits and 14 the most significant. Max sector count is approximately $1872_{10} = Ø750_H$.
15	LINK MAP SECTOR # = Binary values in the range Ø1 to $1A_H$, of the sector holding the first link map of the file.
16	LINK MAP TRACK # = Binary value, in the range ØØ to $4C_H$, of the track holding the first link map of the file.

b. Linkage Map - Sector 1 is used by MDS as the first linkage map sector for the Disk Directory. (See linkage map description below).

LINKAGE MAPS

As files are created by MDS-800, directory entries are made in the directory space and sector linkage maps are created in available (unused) sectors starting with Track Ø sector 24 and progressing sequentially by track and sector through unused sector space. Since MDS-800 allocates file space on a sector-by-sector basis based on what is available in the inactive and unused sector space, quite often sectors used by a file requiring multiple sectors are not allocated adjacent to one another. Therefore, a sector linkage map is created for each file that indicates to the system software which tracks and sectors comprise the entire file. Depending on the size of the file, it may require one or more sectors for

its linkage map. Even if only part of a sector is needed, MDS-800 allocates at least one entire sector for linkage map information per file (unused bytes are filled with ASCII nuls by MDS-800). The format for a linkage map is as follows:

BYTE	CONTENT/USE
1-2	Sector (Byte 1) and track (Byte 2) of any preceeding linkage map sector for the file. Both bytes are nuls if this linkage map sector is the first (or only) sector of the linkage map.
3-4	Sector (Byte 3) and track (Byte 4) of any succeeding linkage map sector for the file. Both bytes are nuls if this linkage map sector is the last (or only) sector of the linkage map.
5-128	Pairs of bytes in the order sector - then - track linking sequential storage areas for the file's contents. Unused bytes are filled with nuls to indicate the end of the linkage and additional sectors (both preceeding and succeeding) are indicated as discussed above. Since Bytes 1-4 are reserved for linking linkage map sectors, 124 bytes are available per linkage map sector for file sector/track pairs allowing 62 such pairs.

As described above, certain sectors on Tracks Ø and 1 (and Sector 1 on Track 2 as you'll see below) are dedicated as Linkage Maps for system information such as the System Initialization Program, Disk Label, and Disk Directory.

TRACK 2

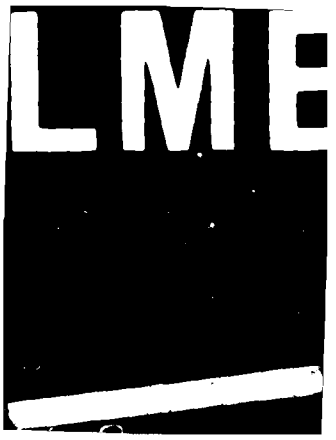
a. Allocation Bit Map - Sectors 2 and 3 are dedicated to the allocation bit map for the diskette. This map is used by MDS-800 to keep track of which sectors on the diskette are being used (i.e., active) versus unused (i.e., inactive or not yet used). One bit is allocated per diskette sector (1 = used, 0 = unused). Since there are 77 tracks at 26 sectors per track, each diskette has a total of 2002 sectors thus requiring 2002 allocation map bits. Two sectors are needed for the allocation bit map since one sector has a capacity of 1024 bits (i.e., 128 bytes/sector times 8 bits/byte.

b. Linkage Map - Sector 1 of Track 2 is dedicated as the linkage map for the allocation bit map described above.

c. File Space - The remaining tracks and sectors on the disk (exclusive of tracks 75 and 76 which are reserved as spares to be used only if some other track is unusable due to wear) are available for user file data. Assignment is made exclusively by MDS-800 or the operating system of any compatible non-MDS system so that the necessary directory entries and linkage maps can be prepared and maintained.

A summary of the track/sector assignments described above is shown in the following table:

TRACK	SECTOR(S)	CONTENT/USE
0	1-23	MDS System Initialization Program
"	24	Linkage Map for MDS System Initialization Program
"	25	Linkage map for Disk Label
"	26	Disk Label
1	1	Linkage map for MDS Disk Directory
"	2-26	MDS Disk Directory
2	1	Linkage map for MDS Allocation Bit Map
2	2,3	MDS allocation Bit Map
"	4-26	Free storage for data files and accompanying linkage maps
and		
3-74	1-26	
75,76	1-26	Tracks reserved as spares to substitute for worn out tracks


LME