PRELIMINARY

Diablo Systems Incorporated

System Operating Procedures Reference Manual

SYSPG03P/R1 JULY 1976

PRELIMINARY

Diablo Systems Incorporated

System Operating Procedures Reference Manual

SYSPG03P/R1 JULY 1976

TABLE OF CONTENTS

TABLE OF CONTENTS (cont)

LIST OF FIGURES

LIST OF TABLES

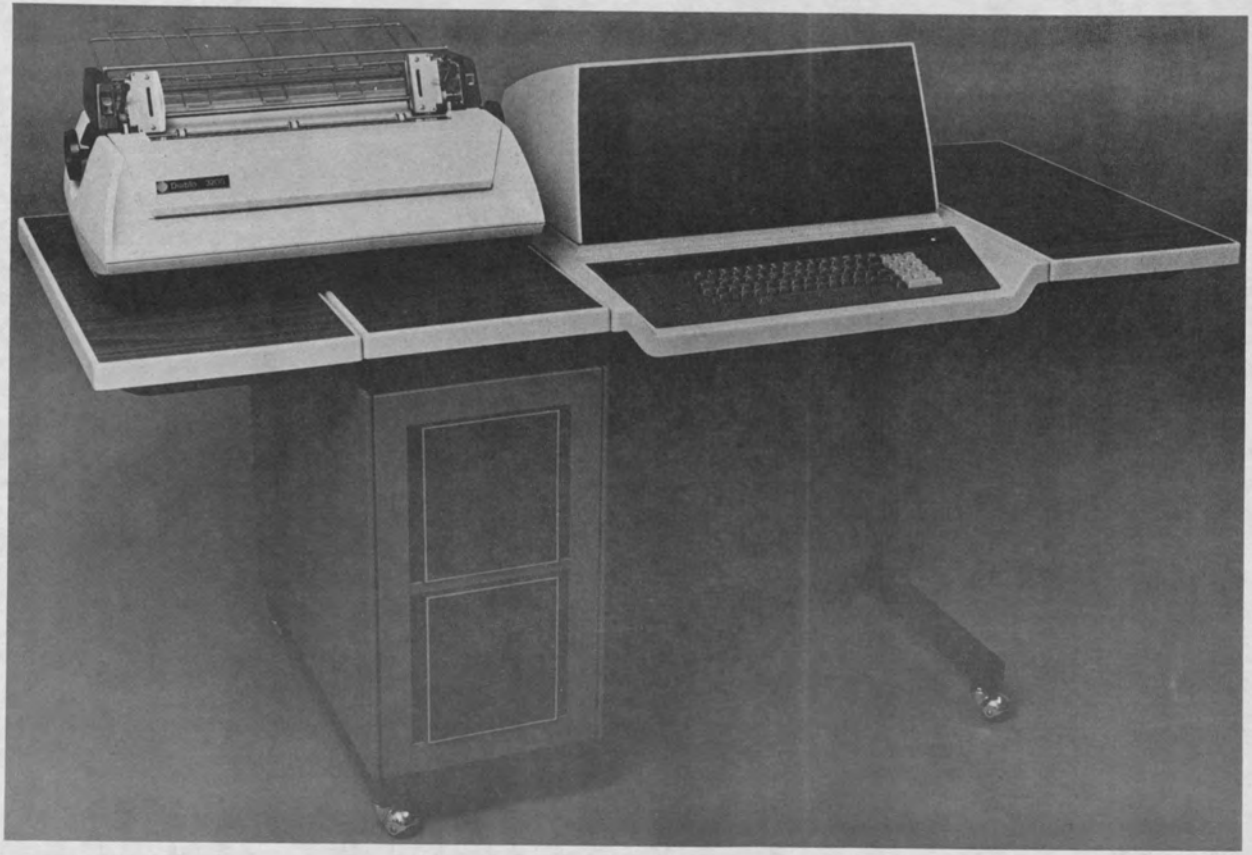


Figure 1-1. System 3200

SECTION I - INTRODUCTION AND GENERAL SYSTEM OPERATION

1.1 SCOPE AND PURPOSE OF MANUAL

This manual is intended for the reader who is familiar with general computer system operation. The purpose of this document is to provide the operator with the general procedure required to perform tasks carried out on the Diablo System 3200. A typical System 3200 is shown in Figure 1-1. Routine tasks performed in the system include those listed below. Each task or group of tasks is discussed in a section of this manual.

1. Applying power, Flexible Disk Operating System (FDOS) initialization, and general program initialization.

2. Creating and editing a file in the system.

3. Compiling and executing a program written in DACL (a high-level Diablo Language).

4. Assembling and executing a program written in assembly language.

5. Operation of system utilities, including the system utility, Sort.

6. Operation of commands used to debug a program.

7. Procedures used to support system operation, replacing printer printwheel and ribbon, interpreting status messages, etc.

1.2 NOTATIONAL CONVENTIONS

Throughout the remainder of this manual, certain notational conventions are followed in the presenting of information. These are listed below.

1. A parenthesized subscript following a string of alphanumeric or numeric characters denote the numbering system to which the characters of the string belong: $F2AC_{(16)}$; hexadecimal system; $125_{(8)}$, octal system; the decimal system is assumed on all strings of digits without a subscript.

2. All italicized words represents variables. For example, the word *register* refers to one of the following symbols from the set of symbols that can be considered a register: A, B, C, D, E, H, L, and M. When the term appears without italics, it is used generically.

3. Capitalized words in a keyboard line entry or a calling sequence represent a keyword, a word with a prescribed meaning to the System 3200 software.

4. Parameters appearing in brackets [parameter] in a keyboard line entry refer to a mandatory input, i.e., a parameter from the set of parameters defined for the fields must appear in the keyboard line entry.

5. Parameters appearing in parenthesis (parameter) in a keyboard line entry refer to an optional input, i.e., a parameter from the set of parameters defined for the field may optionally appear in the keyboard line entry.

6. Either a blank or comma serves as a separator between fields unless otherwise specified.

7. Underlined data in system/operator dialogue indicates an operator response to a system request. The symbol ↵ at the end of a keyboard entry indicates a carriage return.

```
NEW FILENAME?NEWFILE↵
```

SECTION 2 - SYSTEM INITIALIZATION AND GENERAL OPERATING PROCEDURES

2.1 INTRODUCTION

To operate the System 3200 requires a familiarity with the major system components and with the system control panel controls and indicators. Thereafter, two procedures, application of main power and system initialization must complete to permit a program to be executed in the system. Once the system is initialized, a program can be called into execution from the keyboard. Besides understanding these general system operating procedures it is also helpful to understand the concept of a file within the System 3200, the structure used to maintain files (on diskette) in the system, and how to manipulate these files. These topics are the substance of this section.

NOTE

The reader should read and understand Section 2 before attempting the procedures in the remainder of this manual.

2.2 SYSTEM COMPONENTS

As shown in Figure 2-1, a system comprises several major components, typically, a printer, diskette, CRT, control panel, and keyboard. However, the hardware components comprising a given system are determined by system application.

2.3 CONTROLS AND INDICATORS

The controls and indicators of significance to the operator are shown in Figure 2-2 and briefly described in Table 2-1. The operator controls the dual diskette drives by inserting diskettes into the two receptacles on either of the dual drives and closing the access doors.

CAUTION

Do not open the access doors on either of the dual diskette drives while the computer is writing to or reading from a diskette.

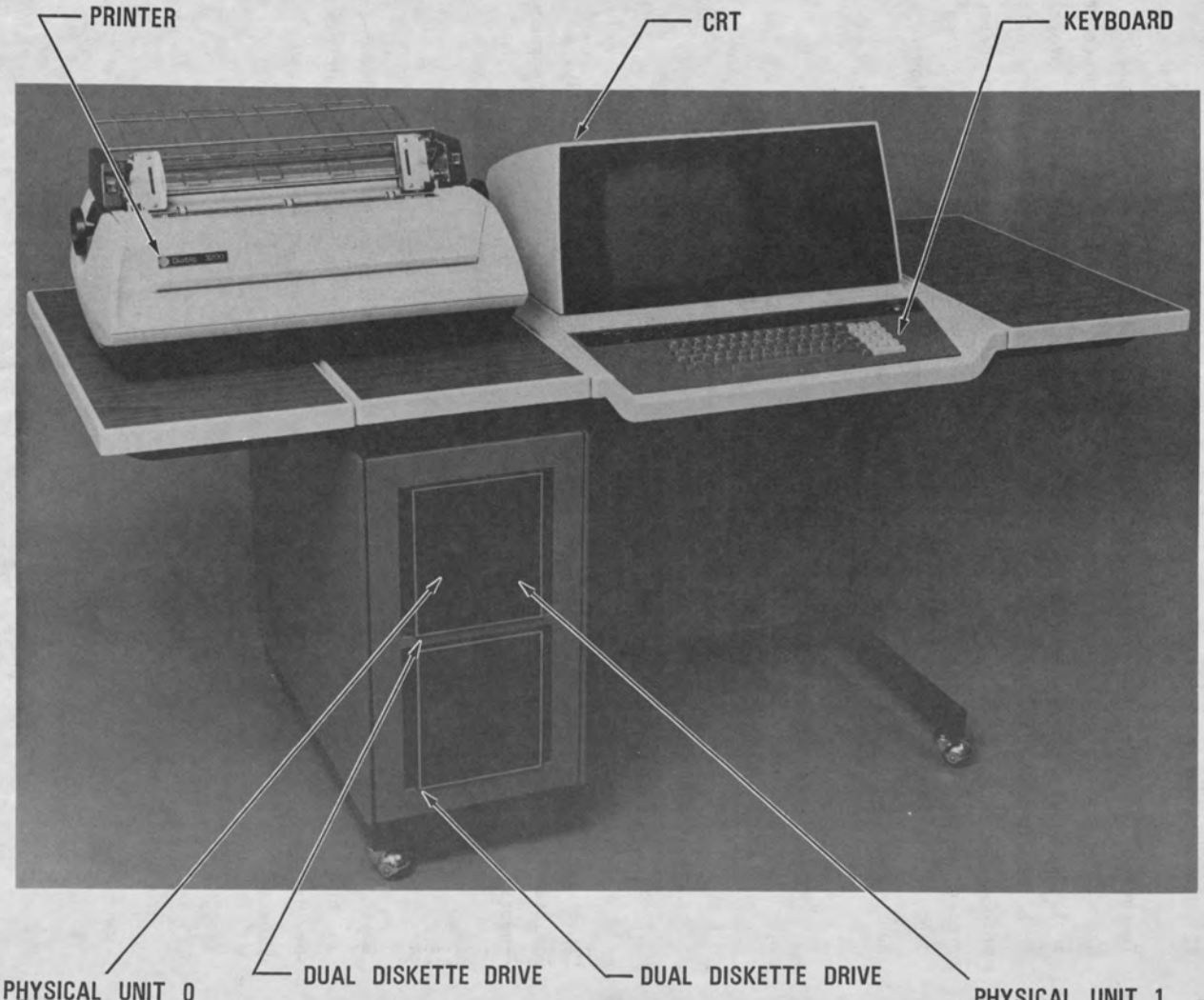
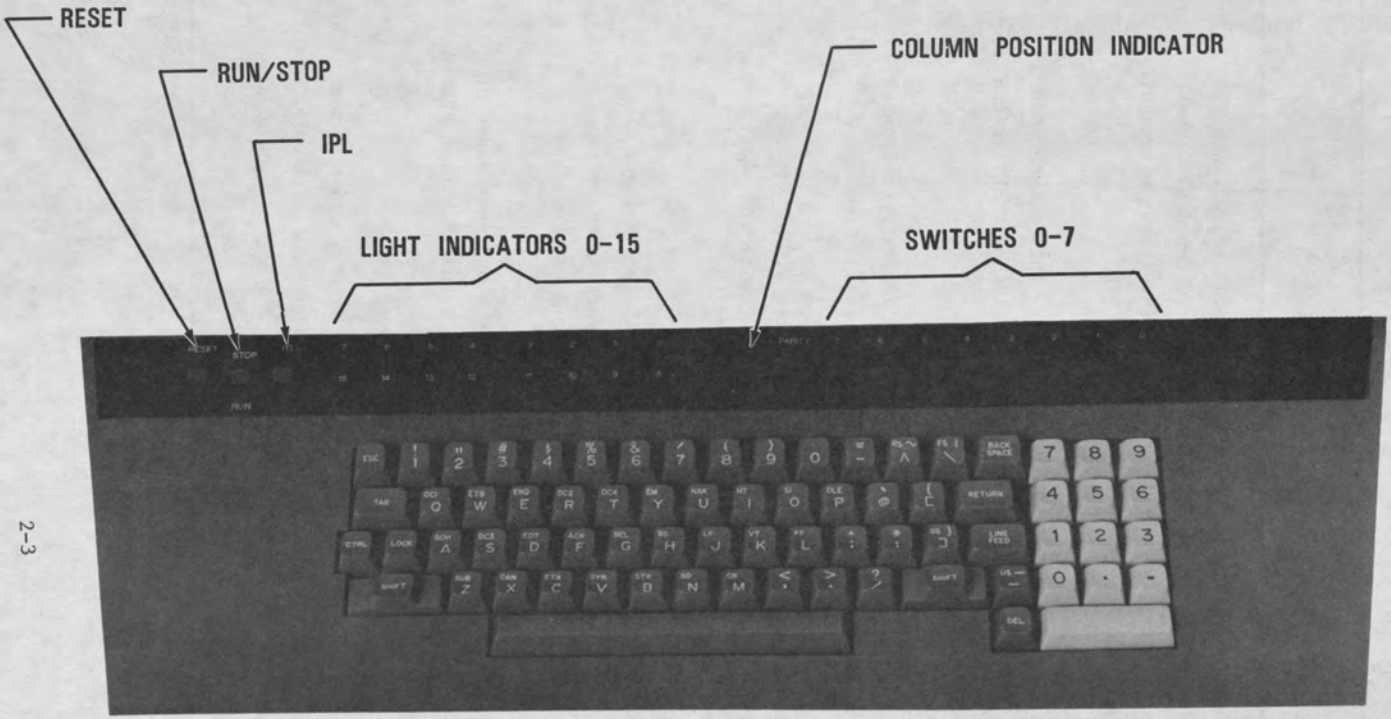


Figure 2-1. System Components



NOTE: Later versions of the keyboard and control panel contain the control panel markings and keyboard layout listed in Table 2-1.

Figure 2-2. System Keyboard and Control Panel

Table 2-1. Control Panel Controls and Indicators

Control/ Indicator	Description
IPL (Initial Program Load)	Causes the system hardware bootstrap to load the contents of the diskette in physical unit 0 (Figure 2-1, top dual diskette drive) in the first 8K bytes of memory. The diskette typically contains the FDOS software for the System 3200.
RUN/STOP	Halts computer operation in STOP position. This switch must be in the STOP position before either the IPL or RESET switch will function.
RESET	Performs a master clear of the computer, causing operation to begin at address 0 with all interrupts disabled.
Column Position Indicator	Indicates the position of the carriage along the print line or cursor horizontal position during operation with FDOS software; indicates status code during IPL.
Light Indicators 3-15	These lights are program controlled and as such reflect the information placed in the lights by the particular program executing in the system at a given moment.
Switches 0-3	These switches, like the light indicators above, are under program control. Thus they can be programmed as required by the system programmer.
SCROLL PTR (Switch 4)	When on, causes the printer to scroll up several lines each time a pause occurs in the transfer of data to the printer (from keyboard or processor). This provides the operator a better view of the data just printed.

Table 2-1. Control Panel Controls and Indicators (Cont)

Control/ Indicator	Description
CRT TO PTR (Switch 5)	When on, causes all data normally destined for the CRT to print on the printer. When off, data is routed to the CRT normally.
STOP CRT (Switch 6)	When on, inhibits the software scrolling feature in the CRT to prevent automatic scrolling when the CRT is sent more than 22 lines of data for display. When this switch is on, a single line can be scrolled by pressing the up/down arrow key on the keyboard or the entire screen can be scrolled by pressing the left/right arrow key on the keyboard.
STOP PTR (Switch 7)	When on, inhibits the printer from printing.
DSK (Light 0)	When on, indicates that the diskette is active.
PTR (Light 1)	When on, indicates that the printer is active.
KBD (Light 2)	When on, indicates that the keyboard is active.

2.4 APPLICATION OF MAIN POWER

Main power is applied to the System 3200 by a power on/off switch mounted to the cabinet front panel, beneath the keyboard.

2.5 SYSTEM INITIALIZATION

Provided with the system are diskette containing the Flexible Disk Operating System (FDOS). The entire FDOS is written in object code on one diskette.

Initialize the system by placing the diskette containing the FDOS software in the logical unit 0 diskette drive receptacle and closing the access door. The diskette should be placed in the receptacle with the diskette label facing the right side of the system and the oblong read/write head slot aligned horizontal and nearest the diskette receptacle. Thereafter, press the control panel STOP switch and then the IPL switch. Set the STOP CRT (switch 6) on to have a memory test executed during IPL.

Next press the control panel RUN switch and observe that the following sequence of numbers are displayed in the control panel 3-digit column position indicator: 999, 901, through 904, 921, 970 through 973, 950, 940, 1 and 3. Next observe that the KBD indicator lights and the following message appears on the system CRT or printer.

CP1.2 REV*xx* - *yy*K SYSTEM

where *xx* is a 2-digit number indicating the revision level of CP1.2 (Control Program 1.2) and *yy* is a 2-digit number specifying the size of the memory contained in the system.

If the system is equipped with more than 20K of memory, the number sequence 901 through 904 is extended one number for each 4K increase in memory size. Thus a 32K system will display 901 through 907. If the STOP PTR switch (switch 7) is off, the 901 through 9*xx* sequence is omitted.

2.5.1 Error Indications

If an error occurs during IPL, the system halts with one of the numbers from the number sequence in the 3-digit column position indicator. The significance of each of these numbers if an error halt occurs is detailed in Table 2-2.

2.5.2 Error Recovery

To correct an error halt condition displaying 999, 900 - 915, and 921, press the control panel RESET switch and attempt the IPL once again.

If the 970 indication is displayed, ensure that the diskette drive access door is properly closed and that the diskette has been inserted in the diskette receptacle properly. Thereafter, attempt the IPL once again.

If the 971 or 974 indication is displayed, attempt the IPL load with another diskette containing the FDOS software. If the error persists attempt the load on another diskette drive. This can be effected by setting switch 1 on to load from logical unit 1, switch 2 on to load from logical unit 2, and both switches 1 and 2 on to load from logical unit 3.

If the 972 indication is displayed, attempt the IPL load on another diskette drive as described for the 971 error indication.

If the 973 indication is displayed, attempt the IPL load with another diskette containing the FDOS software.

If error indications 84*x* and 7*xx* are displayed, press the control panel RESET switch and perform the IPL once again.

If error indications 601 through 607 are displayed, set the PTR OFF switch to the ON position and correct the indicated error. Error indication 604 clears itself when PTR OFF is turned on.

In any error condition persists, contact maintenance personnel to correct the problem.

Table 2-2. IPL Error Indications

Number	Explanation
0	Processor not running.
999 with display in lights 0 - 15.	The bootstrap routine in read-only memory (ROM) which performs the IPL is moved from ROM into random-access memory (RAM) while the number 999 is displayed. The system halting with 999 displayed and an unknown display in light indicators 0 - 15 indicates that a failure occurred during this movement. If the system halts with lights 0 - 15 empty, the routine moved from ROM to RAM successfully; however, failed to initiate properly.
900 - 915 with display in lights 0 - 15.	Indicates an error during execution of the memory test. Light indicators 0 - 7 contain the bit pattern that failed in the memory test while lights 8 - 15 contains the high order eight bits of the address where the failure occurred.
921	Indicates an error during execution of the direct memory access memory refresh phase of the memory test. Light indicators 0-7 contain the bit pattern that failed while light 8-15 contain the high order eight bits of the address where the pattern failed.
970	Indicates that the diskette drive containing the diskette to be loaded is not ready: access door not completely closed, diskette not in receptacle properly, or a hardware failure. Lights 8 - 15 contain the diskette N1 status word. See Appendix B for a description of this word.
971	Indicates that the bootstrap routine is unable to locate track 0 on the diskette. If light indicators 0 - 7 are empty, lights 8 - 15 contain the diskette N1 status word. See Appendix B for a description of the word. If lights 0 - 7 contain data, the data contained is the N0 status word while lights 8 - 15 contain the N2 status word. See Appendix B for a description of these two words.

Table 2-2. IPL Error Indications (cont)

Number	Explanation
972	The bootstrap routine has issued a step command to move the diskette read/write head off of track 0 and onto another track; however, the command is not being carried out by the hardware. Light indicators 0 - 7 contain the N0 status word while lights 8 - 15 contain the N1 status word. See Appendix B for a description of these two words.
973	The bootstrap routine attempted to read a sector from the diskette in which it expected to find binary data but did not. This error condition could be caused by attempting to read a diskette which does not contain FDOS.
974	The bootstrap routine attempted to read a sector from the diskette and found the checksum it calculated was different than that stored on the diskette for the sector. This could indicate that the diskette itself is bad or that the drive is malfunctioning.
84*x*	Any error indication in the 840 through 847 range indicates an unknown interrupt error. The error can occur if the system CRT has failed.
7*xx*	Any 700 error indicates a parity error. The *xx* can assume a value from 0 through 15 which corresponds to the 4K section of memory in which the parity error occurred.
601	The printer is out of ribbon or its cover is open
602	The printer is out of paper.
603	Both the 601 and 602 conditions exist in the printer.
604	The printer is in a printer-check condition, i.e, it is up against a stop or cannot execute the operation it has been commanded to perform.

Table 2-2. IPL Error Indications (cont)

Number	Explanation
605	Both the 601 and 604 conditions exists in the printer.
606	Both the 602 and 604 conditions exists in the printer.
607	The 601, 602, and 604 conditions exists in the printer.

2.6 KEYBOARD OPERATION

Once the system is initialized, most operator functions occur at the keyboard. Thus before proceeding further, some general information on keyboard operation is in order. After the system is initialized, the FDOS Command Processor executes in the computer. Command Processor loads and executes programs on command from the operator. The Command Processor is a point of departure and common return for all system operation. A return to Command Processor is indicated by a colon being printed or displayed.

2.6.1 Calling Disk and Memory Resident Programs into Execution

Once the system has been initialized, programs can be called into execution from the keyboard. This is accomplished by inserting the diskette containing the program to be called into an unused diskette drive and typing in the program name followed by the number of the logical unit containing the diskette. When no unit number is entered by the operator, logical unit 0 is assumed. Logical units are set to the equivalent physical units after initialization. They can be changed by typing: UNIT,*x*,*y*,*z*,... where *x*,*y*,*z*,... are logical unit numbers. This will set physical unit 0 to logical unit x, physical unit 1 to logical unit y, physical unit 2 to logical unit z, etc. To call a program into execution, perform the following keyboard line entry.

[*program or filename*](/*unit no.*)(,*parameter 1...*,*parameter n*)↵

NOTE

The *program or filename*/*unit no.* convention is recognized by the command processor anywhere in a keyboard line entry. Thus if a parameter in a keyboard line entry is a filename, the unit containing the file can also be input when the filename is input.

Variable	Definition
Program name	The name of the program being called.
/	A mandatory delimiter if unit number is specified, not required otherwise.
unit no.	The logical unit containing the diskette that contains the program being called. Default is to logical unit 0.
parameter 1,... *parameter n*	The string of parameters required for the program to execute properly.

Following is an example of a program call containing all the required parameters.

```
COPY,1STFILE/0,2NDFILE/1↵
```

After this call is made, the system copies a file called 1STFILE on unit 0 into a file called 2NDFILE on unit 1. Following is an example of a program call in which the operator is prompted.

```
COPY
↵
```

The system assumes logical unit 0 thus no request for logical unit number occurs. After this call is made, the following is printed:

```
ENTER:  SOURCE FILENAME(S),DEST FILENAME(,OPT)

     OPT ARE:

         C=0(CLOSE AT EOD)    C=1(CLOSE AT EOE)
         C=2(TRUNK AT EOD)
         P=1(PROTECT)         A=1(APPEND)
```

To this the operator responds with one of the options listed as follows:

```
1STFILE/0,2NDFILE/1,C=0,P=0,A=0
```

After this entry is made, the system copies the file called 1STFILE on unit 0 into the file called 2NDFILE on unit 1. The option following the two filenames specify how 1STFILE is to be placed into 2NDFILE. (See Section 6 for more on COPY program.)

2.6.2 Command Processor Status Information to Operator

Command Processor provides a variety of status messages to the operator during system operation. A list is provided in appendix B. One message is important to the operator when calling a program into execution. This message is listed below. The message indicates that an error occurred while command processor attempted to load a program.

LOAD ERR *xx*

Variable	Definition
xx = $02_{(16)}$	Logical unit selected not ready.
$03_{(16)}$,$15_{(16)}$	Disk input/output error occurred during load.
$04_{(16)}$	Disk may not have been initialized.
$0A_{(16)}$	Program not found in directory of diskette in logical unit specified.
$14_{(16)}$	End of file occurred prematurely.
$16_{(16)}$	Block count error exists on diskette specified.
$17_{(16)}$	Block type error exists on diskette specified.
$18_{(16)}$	A checksum error occurred while reading program from diskette.

Corrective measures for these indications are contained in appendix B.

2.6.3 Correcting Typing Errors at the Keyboard

During keyboard line entry, typing errors can be corrected in two ways. If a single character is typed incorrectly, typing DEL (delete) deletes the character from the line entry and causes the deleted character to be overstruck with a dash on the printer or erased on the CRT. Continually typing DEL deletes the characters in reverse order, until part or all characters of a given line entry have been deleted. If an entire line is in error, the line may be deleted by pressing the CTRL (control) key and typing x.

2.6.4 Aborting a Program Call or Keyboard Entry

If it becomes necessary to abort a program call or keyboard entry and return to Command Processor, press the ESC key on the keyboard and observe that a colon prints or appears on the CRT indicating that control has returned to Command Processor. If this does not occur, reinitialize the system.

2.7 FILE STRUCTURE

In the System 3200 all information is stored in files. A file can contain data or a program. Files are maintained on diskette and are read into the system as needed. To understand how a file is maintained on diskette and accessed by the system when needed, it is helpful to show the process involved in preparing a blank diskette for use in the system and thereafter to show the contents of a diskette containing files.

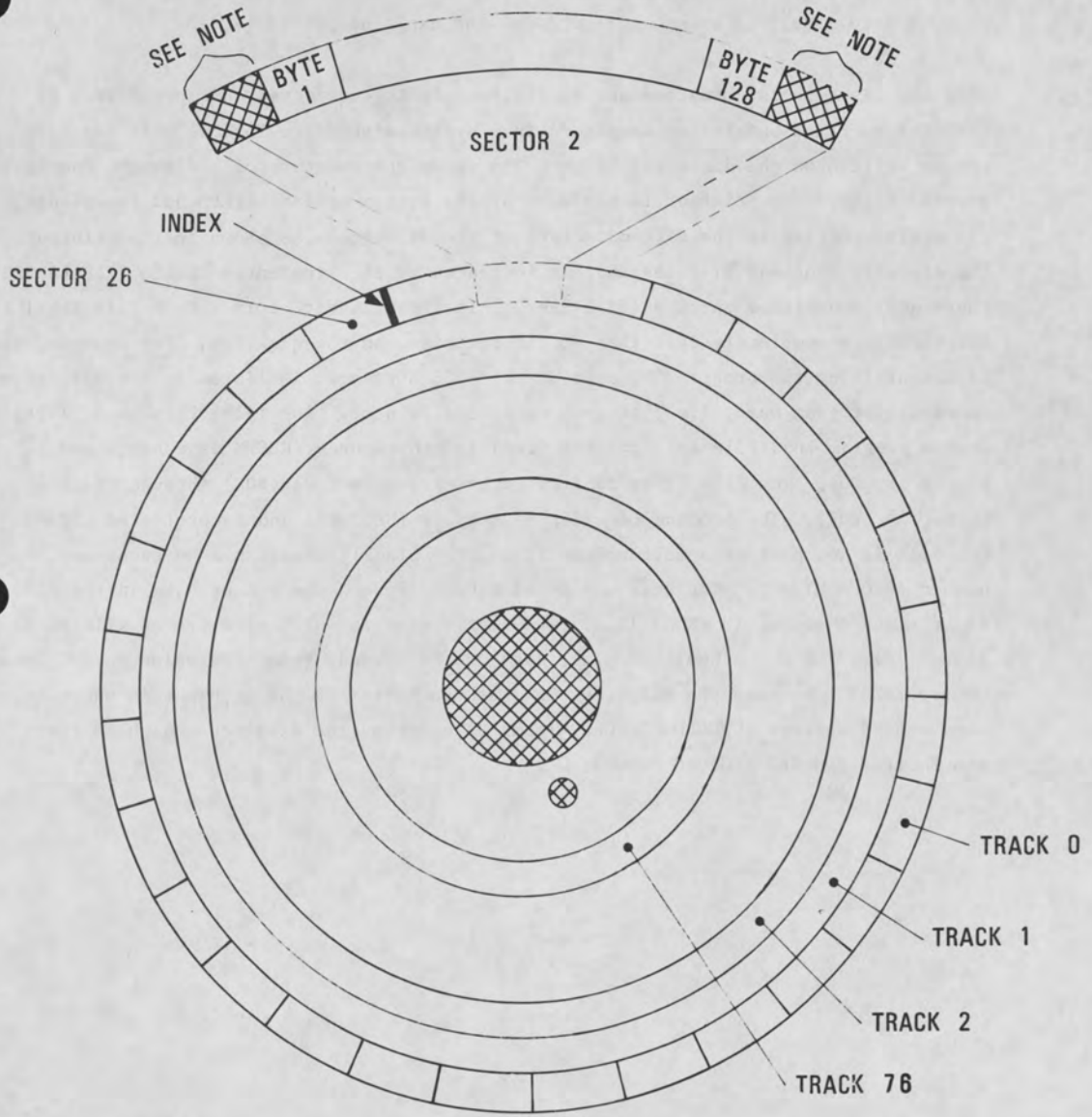
For a diskette to be usable in the System 3200, it must be formatted and initialized. Two system utilities in the FDOS software are available to perform both of these functions. See Appendix C or Section 6.

NOTE

> The terms "blank" and "empty" have limited definitions when used to describe diskette structure. A blank diskette is an unformatted and uninitialized diskette, whereas an empty diskette is formatted and initialized but does not contain any files.

Figure 2-3 shows the structure applied to a blank diskette when the diskette is formatted. Formatting configures the blank diskette into 77 tracks, each track containing 26 sectors and each sector containing 128 bytes.

After the tracks and sectors of a diskette have been formatted, it then becomes necessary to create a file directory on the diskette. This is accomplished by



NOTE:
SECTOR PREAMBLE AND POSTAMBLE CONTAIN HARDWARE DATA.

Figure 2-3. Formatted Diskette

initializing the diskette. When the operator initializes a diskette, sectors 8 through 26 of track 0 and sectors 1 through 26 of track 1 are initialized as the diskette directory. The data contained in each sector of the directory, referred to as a file label, is shown in Figure 2-4 and explained in Table 2-3.

Once the diskette is formatted and initialized it is considered an empty disk. It contains no files but it can be placed in a system disk drive logical unit and data can be written on the diskette. Figure 2-5 shows the contents of a diskette containing several files. The printout is produced by the system utility FILES and represents the active entries in the file directory of the diskette. As shown in the printout, the diskette contains 21 files and, as indicated by the line entry UNUSED FILE LABELS, there are 24 unfilled entries (file labels) in the file directory. Each file label in the directory completely describes a file contained on the diskette. For example, the file containing the program CP, part of the FDOS software, is listed in the directory under its program name, i.e., program name is file name. The file CP is an S (TYPE), system program load file and contains fixed length records (RECFM is a blank and RECLEN is 000). The file CP is located in track 2 sector 01 (BOE) through track 4 sector 15 (EOE). The program occupies 67 sectors (SECTORS) and is protected (FP=P). The file is not part of a multivolume file (MV = blank) thus it has no sequence number (SEQ = blank). The file was created 07/02/76 and the end of data in the file is at track 4 sector 16 (EOD) (i.e., the next sector in which data can be written is sector 16). EOD shows the next place data can be stored; thus, the value in EOD appears one sector larger than the value in EOE. At the bottom of the printout is shown how many unused sectors (UNUSED SECTORS - NUMBER) exist on the diskette and where they are located (UNUSED SECTORS - BOE).

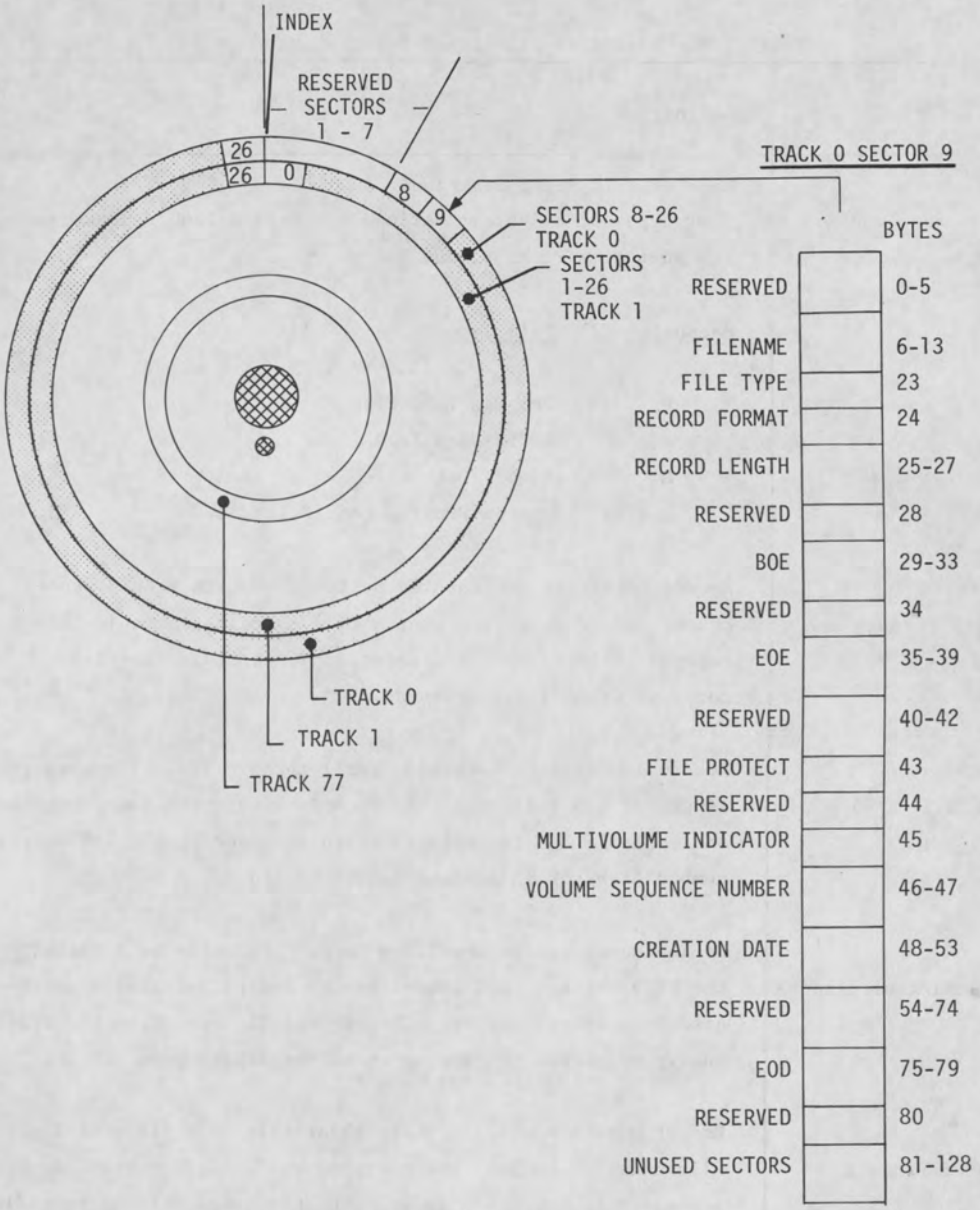


Figure 2-4. Initialized Diskette

Table 2-3. Directory File Label Descriptors

Descriptor	Definition
NAME	Name of file being described.
TYPE	Four types of files are defined for System 3200 represented by the mnemonics P, A, B, and S: Mnemonic — File Type P — Program Load File A — ASCII Data File B — Binary Data File S — System Program Load File
RECFM (Record Format)	Record format is represented by two mnemonics V or a space character. V specifies that a file contains variable length records while a space character indicates that the file contains fixed length records.
RECLEN (Record Length)	Record length is a variable from 1 through 128. A record length of 000 indicates that records within the file described are variable length while a variable between 1 and 128 states number of bytes in a fixed-length record.
BOE (Beginning of extent)	Beginning of extent specifies where physically on a diskette the file begins. BOE comprises two 2-digit variables separated by a space: xx yy. The xx variable specifies the track and yy specifies the sector where the file begins.
EOE (End of Extent)	End of extent specifies where physically on a diskette the file ends. Like BOE, EOE contains two 2-digit variables separated by a space: xx yy. The first variable xx specifies

Table 2-3. Directory File Label Descriptors (cont)

Descriptor	Definition
	the track while the second yy specifies the sector. Refer to the BOE for a description of track and sector.
SECTORS	Sectors specifies the number of sectors in a file.
FP (File Protect)	File protect specifies whether or not a file is write-protected from being written over inadvertently. The mnemonic P in this field indicates that writing in the file is prohibited, while a space character (indicated by a blank) indicates that the file is not protected.
MV (Multivolume)	This descriptor indicates whether or not the entire file is on this diskette. a space character (indicated by a blank) specifies that the file is wholly contained on the present diskette, while the mnemonic C specifies that the file is continued on another file, and, finally, the mnemonic L specifies that the file is multivolume and that this diskette contains the last of the multivolume file.
SEQ (Sequence)	Sequence is a 2-digit variable, 00-99, which identifies a given diskette in a multivolume sequence. Thus a 02 in this field would indicate that this diskette was the second diskette in a multivolume file.
CREATED	This descriptor specifies the date a file is created: month, day, and year.
EOD (End-of-Data)	This descriptor specifies the last sector of a file that contains data. Like BOE and EOE, EOD comprises two 2-digit variables separated by a space: xx yy. The xx variable specifies track while yy specifies sector. The track and sector specified is the location of the next unused sector in the file.

Table 2-3. Directory File Label Descriptors (cont)

Descriptor	Definition
UNUSED SECTORS	If a file is not completely filled with data the number of empty sectors remaining in the file is shown in this field.

```
ENTER COMMAND (LST,FLS,ALO,DEL,TRU,REN,PRO,UNP,TYP,LBL,DAT,UNT)
LST

FILES ON UNIT 0  VOL SYSD15                                    MASTER

  NAME    TYP RF RLN  BOE    EOE   SECS  FP MV SEQ  CREATED    EOD  UNUSED SECS

CP          S     000 02 01 04 15    67   P        07/02/76  04 16          0
FILES       S     000 26 06 27 16    37   P        07/13/76  27 17          0
SAVE        S     000 10 09 11 06    24   P        07/07/76  11 07          0
COPY        S     000 09 19 10 08    16   P        07/07/76  10 09          0
FORMAT      S     000 11 07 11 18    12   P        07/07/76  11 19          0
INIT        S     000 11 19 11 22     4   P        07/07/76  11 23          0
TIME        S     000 04 16 04 18     3   P        07/07/76  04 19          0
DUMP        S     000 13 08 13 16     9   P        07/07/76  13 17          0
PRINT       S     000 13 17 14 10    20   P        07/07/76  14 11          0
DISKCOPY    S     000 14 11 14 17     7   P        07/07/76  14 18          0
FIXD        S     000 25 11 25 24    14   P        07/12/76  25 25          0
ASR         S     000 04 19 05 24    32   P        07/08/76  05 25          0
@           S     000 15 05 15 16    12   P        07/08/76  15 17          0
UPD         S     000 15 17 17 20    56   P        07/08/76  17 21          0
ASEM        S     000 17 21 19 11    43   P        07/08/76  19 12          0
XREF        S     000 21 25 22 13    15   P        07/09/76  22 14          0
DACL        S     000 20 01 21 24    50   P        07/09/76  21 25          0
INT         S     000 22 14 24 06    45   P        07/09/76  24 07          0
EXOR        S     000 24 07 24 20    14   P        07/09/76  24 21          0
COMPAT      S     000 25 25 26 05     7   P        07/12/76  26 06          0
MEMTST      S     000 24 22 25 10    15   P        07/09/76  25 11          0

UNUSED FILE NAMES  24

UNUSED SECTORS
 BOE    NUMBER
05 25      98
11 23      37
14 18      13
19 12      15
24 21       1
27 17    1284
ENTER COMMAND
```

Figure 2-5. Active Diskette Directory File Labels

SECTION III - CREATING AND EDITING FILES

3.1 INTRODUCTION

In the last section, the FDOS software was loaded into the system. This section describes how to create a file on diskette and, once created, how to edit the file on diskette.

3.2 FILE CREATION

File creation involves preparing an empty or partially filled diskette to contain a new file and the storing of data into the file once it has been prepared.

3.2.1 Preparing a Diskette to Contain a New File

Once the operator has decided on which diskette to place the new file, the next task is to allocate sufficient space on the diskette to contain the file. To allocate space for a file, the operator issues an allocate-file command to the utility program FILES. FILES requests from the operator the proposed name, file type, and record format to be assigned to the file in the file directory. In addition, FILES requests from the operator the number of sectors to be allotted to the file. To reply to this request, the operator must estimate the probable amount of space required to store the final contents of the file. (Some guidelines are provided in the following procedure.) See Section 6 for more on FILES.

To allocate space for a new file perform the following:

(1) Initialize the system.

(2) Place the diskette to contain the new file, if other than the diskette used to initialize the system, in a logical unit other than that containing the diskette used to initialize the system.

(3) Call the FILES program by typing the following:

FILES/*unit*,*unit*↵

The first variable must specify the logical unit containing FILES. The second variable must specify the logical unit containing the diskette in which the new file will be allocated.

(4) Observe that the following message appears on the printer or CRT.

ENTER COMMAND (LST,FLS,ALO,DEL,TRU,REN,PRO,UNP,TYP,LBL,DAT,UNT)

(5) Type in the following command followed by a carriage return:

ALO *filename*↵ where *filename* is the name of the file being allocated.

(6) Observe that the following message appears on the printer or CRT.

SECTORS? (NO ENTRY = MAX AVAILABLE)

(7) Enter the number of sectors that the file will ultimately contain. As a general rule 25 sectors can hold approximately four pages of printout. It is also easier to allocate more space than might ultimately be required. If no entry is made, the maximum number of available sectors on the diskette will be allocated to the file. Enter the number of sectors as follows:

SECTORS? (NO ENTRY = MAX AVAILABLE)*nnnn*↵

where parameter *nnnn* is a number between 1 and 1925.

NOTE

It is possible, if a large number of sectors is being allocated to a file that the number specified by the operator is larger than the maximum available on the diskette. If this is the case, a message to this effect will appear on the printer or CRT:

NOT ENOUGH ROOM TO ALLOCATE *filename* ON UNIT *n* VOL *volume id*

The operator should use a diskette with more space or reduce the number of sectors requested.

(8) Observe that the following message appears on the printer or CRT:

TYPE? (A,B,P,S)

(9) Enter the letter (A,B,P, or S) that describes the type of data that will be entered into the file (P = program, B = binary data, A = ASCII data, or S = system program) as follows:

TYPE? (A,B,P,S)*x*↵

where parameter *x* is one of the four letters listed.

(10) Observe that the following message appears on the printer or CRT:

RECORD FORMAT? (V, SPACE)

(11) Enter the letter V or press the space bar on the keyboard to select either a variable length record (V) or a fixed length secord (space). Enter the selected character as follows:

RECORD FORMAT? (V, SPACE)*x*↵

where parameter *x* is one of the two characters listed.

(12) If a variable length record is selected in step 11, proceed to step 13; otherwise, observe that the following message appears on the printer or CRT.

RECORD LENGTH? (1-255)

Record length is important when fixed length record files are to be created. Fixed length records are typically used in random access data files. If a fixed length record is selected in step 11, select an appropriate record length in the range specified and enter the quantity followed by a carriage return.

(13) Observe that the following message appears on the printer or CRT:

FILE *filename* ALLOCATED ON UNIT *n* VOL *volume id*
BOE *xx yy* - EOE *xx yy*

where *xx* = track number, *yy* = sector number.

This message indicates that a file has been allocated and describes the location, the name of the file, and the unit containing the diskette.

(14) Repeat the procedure to create two or three additional files for later use in editing a file.

3.2.2 Entering Data into a File

To insert data into a file, the operator calls system program EDIT, informs EDIT of the name of the allocated file to be filled with information and thereafter enters the specified data into the file. This procedure is described in the remainder of this section by first providing the necessary commands and parameters required and then by providing an example of a program being written into an allocated file.

3.2.2.1 EDIT Call Statement

To enter data into a file call EDIT by typing the following:

[EDIT][␣][,*source file*][,z][,N]↵

Line Entry Item	Explanation
EDIT	Command Word identifying the EDIT software module.
␣,	A space followed by a comma must follow the command word EDIT.
source file	The name of the file where data is to be stored. If not already allocated, EDIT will allocate the file.
z	The parameter z can assume three forms, the letters, A, D, or T. If A is specified EDIT assumes that the operator intends to create an assembly language source file. A D specifies a DACL (Diablo Application Compiler Language) source file, while a T specifies a text file, i.e., a data file or a language text file.
N	Indicates that the file contained in the call statement is to be a new file.

Observe that the following message appears on the printer or CRT in response to the call statement.

```
*DO NOT REMOVE SOURCE MEDIA**
    ***EDITOR 1.1***
 variable EDITOR READY
```

Where *variable* is determined by what was input in the call statement for the parameter z: A produces ASSEMBLER; D, DACL LAN and T, TEXT.

CAUTION

Do not remove or change diskette during editing operation.

NOTE

If an improper value is entered during the call for any parameter other than the command EDIT, the EDIT software module executes with the erroneous data or prints an error message. If an error message is displayed/ printed, EDIT will also prompt the operator for the correct parameter. The messages EDIT prints or displays described in paragraph 3.6.

If the correct message does not appear, abort the call by pressing the ESC key and enter the complete EDIT call statement once more.

If the correct message appears, EDIT is ready to accept data entered at the keyboard. However, before entering data into the file, refer to paragraph 3.4 for information on operation of the software module EDIT.

3.2.2.2 EDIT Call Using EDIT Prompting

If the EDIT call statement is made without all line entry items included in the line entry, EDIT will prompt the operator for those parameters it requires to execute. To call EDIT without supplying all parameters in the initial call statement, refer to paragraph 3.6.

3.3 EDITING AN EXISTING FILE

Editing an existing file is similar to creating a new file. Both processes create a new file. However, during the editing of a file some or all of the data contained

in an existing file is moved into a previously allocated, newly created file along with insertions, changes, corrections, or deletions introduced at the keyboard. In some cases, the editing process affects only the newly created file, i.e., the existing file being edited is unchanged during the editing process. In other instances, however, both files are changed during the editing process.

Call EDIT to edit an existing file by typing the following:

[EDIT][␣][*source filename*][,*dest. filename*][,z][,y]↵

Line Entry Item	Description
EDIT	Command which calls EDIT software.
␣	Mandatory space.
source filename	The name of the file being edited.
dest. filename	The name of the file being created to contain the edited version of the existing source file.
z	Specifies the type of language contained on the file being edited. A = Assembly Language D = DACL Language T = Text (a data file containing data or English text.)
y	Six parameters are possible in this field: X, M, S, C, N, E. Parameters X, M, and S are similar, however, different from C, N, and E. C - causes the source file to be transferred intact into the newly created destination file. N - used for file creation, this parameter indicates to EDIT that a new file is being created.

E - indicates to EDIT that more data will be entered onto the end of the existing file specified in the call statement. When E is used, the call statement contains only one filename, that of the file in which more data is to be added.

X - Multi-pass option; output on last file used.

M - Multi-pass option; output to designated file.

S - Single-pass option; output to designated file.

NOTE

The pass option defined by parameters X, M, or S applies only to the find-string, locate-string, find-last, locate last and locate and modify instructions. The X or M option permits a multiple pass of both source and destination files to occur. A pass occurs when a command such as locate-string is input at the keyboard when half of the source file has been edited and transferred into the destination file. The locate-and-modify command with X or M selected causes EDIT to search the remainder of the source file in search of the string specified in the locate-and-modify command and thereafter to search through the destination file up to the position in the destination file where the locate-and-modify command was issued. Once this pass has completed, issuing any of the five commands listed previously produces another pass with the destination file being searched first followed by the source file. At the completion of the editing operation, with the X option selected the final edited information would appear in the last file in which a pass completed. With M selected the final edited information, if the last pass did not complete in the destination file, would be transferred into the destination file specified in the call. An S would not permit a pass to move beyond the source file.

After the call to edit a file is issued, observe that the following message appears on the printer or CRT:

```
*DO NOT REMOVE SOURCE MEDIA**
    ***EDITOR 1.1***
 variable EDITOR READY
```

Variable is determined by what was input in the call statement for the parameter z A produces ASSEMBLER: D, DACL LAN: and T, TEXT.

If the correct message does not appear in response to the call statement, abort the call by pressing the ESC key and enter the EDIT call once more.

If the correct message appears, EDIT is ready to accept data entered at the keyboard. However, before entering data into the file, refer to paragraph 3.4 for information on operation of the software module EDIT.

3.4 EDITOR OPERATION

The operation of EDIT requires an understanding of what constitutes an input of data to a file being created or edited and what constitutes a command to EDIT. It is also important that the operator understand the buffering operation that EDIT carries out throughout the course of its operation. Finally, it is essential that the operator understand what occurs when EDIT is terminated either normally or abnormally. The following paragraphs will address each of these topics. In addition, EDIT commands are listed in this section as a convenience to the operator. More information on EDIT is contained in the System 3200 Editor Operation Reference Manual.

3.4.1 EDIT Keyboard Operation

Immediately after the "*variable* EDITOR READY" message appears on CRT or printer, the appropriate device performs a carriage return and displays a "?" to inform the operator the next keyboard input will be processed by EDIT. All editor commands begin with a colon (:), thus anything typed which is not preceded by a colon is entered into a line of the file. Besides distinguishing between data and command inputs, the operator must be careful about typing nonprinting characters such as backspace or any of the characters typed with the CTRL key held down. These appear nowhere in the printout of a file yet exist physically within the file. The existence of nonprinting characters in a file will cause errors when the file is assembled or compiled. The nonprinting characters tab and space are an exception to this general rule. Tab or space have specific functions when creating a file for the assembler or DACL. Typing tab or space after a "?" appears causes a tabulation to column 10 of the print or display line. Thereafter, either key produces a single space.

Typing errors can be deleted by typing the DEL key or typing x with the CTRL key pressed. Typing DEL causes a backspace of one character; the character originally occupying the space is deleted. CTRL x deletes all characters back to the last tab stop.

3.4.1.1 Keyboard Vertical Arrows Key

The vertical arrows key is marked with an up-pointing and down-pointing arrow. Pressing the key during EDIT operation on a CRT system moves the EDIT line pointer down. Holding the CTRL key down while pressing the key moves the EDIT line pointer up.

3.4.1.2 Keyboard Horizontal Arrows Key

The horizontal arrows key is marked with a right-pointing and left-pointing arrow. Pressing the key during EDIT operation on a CRT system moves the CRT cursor to the right. Holding the CTRL key down while pressing the key moves the CRT cursor to the left. Once the key is pressed, EDIT enters cursor modify mode. In this mode characters may be inserted or deleted in a line. Typing a character causes the character to be inserted to the right of the character underlined by the CRT cursor. Also pressing the DEL key deletes the character above the cursor in this mode and pressing the left slash (\) key terminates a line at the present position of the cursor. The cursor modify mode is in effect, even if the line pointer is moved to another line, until the RETURN key is pressed.

3.4.2 EDIT Buffering Operation

EDIT operates with a buffer which is 21 lines long and 79 characters wide. During file creation, each line entered on the keyboard up to line 21 is maintained in the buffer. After the twenty first line has been entered, each line thereafter enters at the bottom of the buffer while the top line of the buffer moves into the destination file on disk. During the editing of a file, the same buffering operation occurs. Lines from the source file enter the buffer along with data input from the keyboard. When 21 lines have entered the buffer either from the keyboard or the source file, subsequent lines enter at the bottom of the buffer and lines at the top of the buffer are moved into the destination file on disk.

3.4.3 Concluding an EDIT Operation

Operation with EDIT can be terminated in a normal manner or in an abnormal manner. A normal termination involves using one of three EDIT commands: End, End and De-allocate, and End and Delete. These commands are discussed in the System 3200 Editor Operation Reference Manual. For purposes of this discussion, one feature of each of these commands is significant: the manner with which each closes the file containing the edited data. Once all the data has entered the edited file, only the End and De-allocate command de-allocates the remaining unused space of the edited file. Thus, if an edited file initially contained 25 sectors and the editing operation used only 15 sectors, the remaining 10 would be eliminated from the file by the End and De-allocate command. The remaining two commands leave the length of the file containing the edited data undisturbed. Both of these features should be considered by the operator when terminating an EDIT operation. An abnormal termination involves only typing the ESC character. If ESC is pressed before 22 lines have been entered or edited, the empty destination file is closed; however, the destination file, whether allocated by the operator or by EDIT immediately after being called, remains on diskette. If ESC is pressed after 22 lines have been entered, the destination file is closed in the same manner as if the E/ command had been issued.

3.4.4 Summary of Editor Commands

For the convenience of the operator, a summary of EDIT commands is printed in Table 3-1.

Table 3-1. EDIT Command Summary

Command[1]	Description
:A	automatically duplicate pointed line.
:C	copy pointed line to bottom of screen, delete pointed line from original location and position pointer to said location to type a new line
:CD	copy pointed line to bottom of screen temporarily, delete pointed line from original location, position pointer to said location to type a new line, and then delete text of original line from bottom of screen.
:D (*n*)	delete specified line of source file in new file.
:D (*n*,*n*)	delete lines *n* to *n* of source file in new file.
:E	copy source file from current position to end of file and end edit operation.
:E/	end edit file operation at end of current line.
:EF	copy source file from current line to end of file but do not terminate edit program.
:EOD	copy source file from current line to end of file and de-allocate remaining space allotted to file.
:F	begin a copy search at beginning of each line for last string entered to a FIND command.
:F (*string*)	begin a copy search throughout the file for given string positioned at beginning of a line.
:I	reposition cursor to allow typing of a new line between previous pointed line and next line.

[1]Parameters in the table appear to be entered on more than one line to fit the column of the table. The operator should note that the command and all parameters must be input in a single line followed by a carriage return.

Table 3-1. EDIT Command Summary (Continued)

Command[1]	Description
:L	begin a copy search for last string entered to a LOCATE command.
:L (*string*)	begin a copy search through the file for a given string anywhere within a line.
:LM (*old*) (*symbol*)[2] (*new*)	begin a search through the file and execute modification of old text by new text.
:M	repeat last MODIFY command executed.
:M (*old*)<(*new*)	search current line, replace specified old text by new text.
:M (*old*)>(*new*)	search current line, insert specified new text at end of specified old text and copy remainder of line.
:M (*old*)\(*new*)	search current line, insert specified new text at end of specified old text and delete remainder of line.
:M/(*old*) (*symbol*)[2] (*new*)	store the designated old and new text for automatic use with later LOCATE and MODIFY instructions.
:N (*n*)	copy source file and position to line number specified.
:P*	turn off printer, terminating printing of input and output lines.
:PF	turn off printing of output file lines.
:PI	print input file lines when file is read.
:PO	print output file lines when file is written.

[2] *Symbol* can be either <, =, or >.

Table 3-1. EDIT Command Summary (Continued)

Command[1]	Description
:PP	print all lines in the buffer.
:R	erase current or pointed line, and allow typing of a replacement line.
:SA	erase all lines from pointed line to top of screen and buffer.
:SB	erase all lines from pointed line to bottom of screen and buffer.
:SF	skip from current line until last string entered to a SKIP FORWARD command is found or end of file, without copying.
:SF (*string*)	skip from current line until specified string is found or end of file, without copying.
:SK	start a copy search for last string entered to a SKIP STRING command.
:SK (*string*)	skip from beginning of file until string is found or end of file is reached, without copying.
:SP	erase all lines of screen and buffer.
:X (*n*)	skip forward from current position to line specified without copying.
:+ (*n*)	position pointer forward the number of lines specified and print the line when positioned.
:- (*n*)	position pointer backwards the number of lines specified and print the line when positioned.

Table 3-1. EDIT Command Summary (Continued)

Command[1]	Description
:=	print current line.
:#	print current line numbers of source file and scratch file.
:*	print active source file and scratch file names, allowing operator to change source file.
:* (*name/unit*)	change source file to source file specified, bypass printing of current source name.
:$ (*n*)	skip from beginning of file to line number specified without copying.

3.5 EXAMPLE OF FILE CREATION AND EDITING

Included in this paragraph is an example of file creation and file editing. The example is divided into three parts: file allocation, file creation, and file editing (Figure 3-1, 3-2, and 3-3, respectively).

3.5.1 File Allocation (Figure 3-1)

File allocation requires the operator to call the system utility FILES. As shown in Figure 3-1, a call to FILES causes the "ENTER COMMAND..." message to appear. In response to this the operator requests that a file be allocated and given the name EXAMPLE1. FILES responds with a request for the number of sectors to be allocated to the file. The operator requests 25. Next FILES asks the operator for a file type. It has been decided that the file will contain a DACL program thus the operator selects an A (ASCII) file type. Accepting the operator's file type, FILES next requests the proposed format of records that will be contained in the file. To this the operator responds with a V which requests a variable length record.

Receiving the last input from the operator, FILES allocates a file on the diskette in logical unit 0. The space allocated to the file begins on track 27 at sector 11 and ends on track 28 sector 09. Receiving this acknowledgment from FILES, the operator allocates another file entitled EXAMPLE2 for later use during file editing.

3.5.2 File Creation (Figure 3-2)

File creation requires the operator to call the software module EDIT. As shown in Figure 3-2 the call to EDIT for file creation involves typing EDIT followed by a blank and a comma (,). Thereafter, the call requires the name of the file being created followed next by the language to be used in the file: DACL, Assembler, or text. These parameters are represented respectively by EXAMPLE1 and the letter D. The space-comma immediately following EDIT informs EDIT that the source file normally appearing in this position for file editing is missing, and thus that this is a file creation call. The N which concludes the call statement informs EDIT that the file EXAMPLE1 is a new file.

```
:FILES

ENTER COMMAND (LST,FLS,ALO,DEL,TRU,REN,PRO,UNP,TYP,LBL,DAT,UNT)
ALO EXAMPLE1
SECTORS? (NO ENTRY=MAX AVAILABLE)25
TYPE? (A,B,P,S) A
RECORD FORMAT? (V,SPACE) V

FILE EXAMPLE1 ALLOCATED ON UNIT 0   VOL D13
BOE 26 03 - EOE 27 01
ENTER COMMAND



ALO EXAMPLE2
SECTORS? (NO ENTRY=MAX AVAILABLE)25
TYPE? (A,B,P,S) A
RECORD FORMAT? (V,SPACE) V

FILE EXAMPLE2 ALLOCATED ON UNIT 0   VOL D13
BOE 27 02 - EOE 27 26
ENTER COMMAND
```

Figure 3-1. File Allocation

```
:EDIT ,EXAMPLE1,D,N

                    *** EDITOR 1.1 ***

          DACL-LAN EDITOR READY


?NAME     DIML5
?HOURS    FIRM 2.2
?RATE     FIRM 4.2
?TAX      FIRM 4.2
?START    KEYIN *L,*L,*L
?         KEYIN "PAY TO THE ORDER OF ", NAME
?         KEYIN "HOURS WORKED ", HOURS
?         KEYIN "HOURS WORKED ", HOURS
?         KEYIN "RATE PER HOUR N", RATE
?         KEYIN "tAX ", TAS
?         MULT HOURS, RATE
?         MULT RATE, TAX
?         SUB TAX, RATE
?         PRINT "NET EARNINGS = "RATE
?         GO TO START
?:PP


          NAME     DIML5
          HOURS    FIRM 2.2
          RATE     FIRM 4.2
          TAX      FIRM 4.2
          START    KEYIN *L,*L,*L
                   KEYIN "PAY TO THE ORDER OF ", NAME
                   KEYIN "HOURS WORKED ", HOURS
                   KEYIN "HOURS WORKED ", HOURS
                   KEYIN "RATE PER HOUR N", RATE
                   KEYIN "tAX ", TAS
                   MULT HOURS, RATE
                   MULT RATE, TAX
                   SUB TAX, RATE
                   PRINT "NET EARNINGS = "RATE
          >        GO TO START
?:E/

        ** FINAL OUTPUT ON UNIT 0, FILE EXAMPLE1  **
```

Figure 3-2. File Creation

In response to the call statement, EDIT identifies itself by printing/displaying its nam followed by the statement: DACL-LAN EDITOR READY. EDIT is ready to place DACL language statements in file EXAMPLE1. Beneath the ...READY statement EDIT performs several line feeds and types a ? at the beginning of a line. This signals the operator that all inputs to the keyboard will be processed by EDIT. As shown in Figure 3-2, the operator enters 15 DACL statements, some of which contain typing errors, incorrect punctuation, and misspelled words. The sixteenth line, unlike the previous lines begins with a colon (:). This is the preface to an EDIT command and thus prevents the line from being entered into the file. The EDIT command following the colon, PP, causes the content of the new file to be printed/displayed. As mentioned earlier in the general discussion of EDIT, twenty one lines of data input to EDIT are always maintained in a buffer, thus the 15 lines shown in Figure 3-2 exist in the buffer and not physically in file EXAMPLE1. The fifteen lines of data are placed in file EXAMPLE1 when the operator types the command, :E/. Acknowledgment that the fifteen lines of data input to the buffer has entered file EXAMPLE1 is implied by the EDIT message: ** FINAL OUTPUT ON UNIT 0, FILE EXAMPLE1 **.

3.5.3 File Editing (Figure 3-3)

File editing requires the operator to call the software module EDIT also. As shown in Figure 3-3 the call to EDIT for file editing involves typing EDIT followed by a blank. Thereafter, the call requires the name of the file being edited followed next by the name of the file to contain the new edition of the edited file. In Figure 3-3, these parameters are represented by EXAMPLE1 and EXAMPLE2, respectively. Next EDIT requires the language involved in the editing process: DACL, Assembler, or text. This is represented by the selection of D in the example. Finally, EDIT requires an editing option: pass option, copy, or extension. This is represented by the selection of the M pass option in the example.

In response to the call statement, EDIT identifies itself, states that it is ready to EDIT a DACL language file, and, additionally issues a warning not to remove the diskette containing the files used in the editing process. The familiar question mark (?) appear at the start of a line beneath the ...READY statement to inform the

operator that all keyboard inputs will be processed by EDIT. As shown in Figure 3-3, the operator issues a series of commands to EDIT. Seeing an error in the DIM statement at the beginning of the program, the operator commands EDIT to position to line 1, :N1. At line 1, the operator issues the command :M and follows the command with the text presently in line 1 which is to be deleted, DIML5. This is followed by the text to replace that being deleted, DIM 15.

The word FIRM should be FORM in lines 3, 4, and 5. Thus the operator issues the command :LM followed by the text to be located and deleted, FIRM, followed by the text to be inserted in place of that deleted, FORM. EDIT makes one complete pass through EXAMPLE1 in search of the word FIRM. Each place it encounters FIRM it substitutes FORM. During the process, each line in file EXAMPLE1 searched by EDIT beginning with line 2 is moved into file EXAMPLE2. When EDIT reaches the end of file EXAMPLE1, it begins searching in EXAMPLE2 and placing each line searched in file EXAMPLE2 into file EXAMPLE1. Since EDIT began with line 2 in EXAMPLE1, it stops with line 1 in EXAMPLE2. This represents a complete pass. However, now EXAMPLE2 has become the file being edited and file EXAMPLE1 has become the file receiving the newest edited data. Completion of the :LM command is indicated by line 1 being printed/displayed. The :LM command is shown here to illustrate its function; however, its use here is not typical. There are two reasons for this. First, the original source data being edited is altered during the second pass when source and destination files exchange roles. Unless the operator took the precaution of copying the source data into another file before beginning the editing operation, the original source data no longer exists. Secondly, the line numbers established in the source file is intended to aid the operator throughout the editing process yet the second pass of the :LM command destroys the usefulness of these source file line numbers by altering the source file contents.

After EDIT output line 1, the operator issues command :N7. This command causes EDIT to print line 7 of file EXAMPLE1. Command :D8 causes EDIT to delete line 8 of file EXAMPLE1. As seen in the output of the newly created file, line 7 and 8 are identical, mistakenly input twice. The operator next uses :LM to correct an incorrect spelling in line 10 or the original file. During typing of the second :LM command, the operator keys in a comma (,) for a less-than (<) sign. Using the DEL key he deletes the incorrect character and types in the correct one. Following the second :LM command, the operator makes two attempts to change "RATE, to ",RATE.

He succeeds on the second try and thereafter changes GO TO to GOTO. The operator then completes the editing process by requesting an output of the newly edited data followed by a :E/ command which concludes the editing process.

```
:EDIT EXAMPLE1,EXAMPLE2,D,M


                        *** EDITOR 1.1 ***
* DO NOT REMOVE SOURCE MEDIA **
          DACL-LAN EDITOR READY


?:N1
        NAME    DIML5
?:M DIML5 <DIM 15
?:=
        NAME     DIM 15
?:LM FIRM<FORM
        NAME     DIM 15
?:N7
                 KEYIN "HOURS WORKED ", HOURS
?:D8
?:LM "tAX ", TAS<"TAX ", TAX
                 KEYIN "HOURS WORKED ", HOURS
?:N10
                 MULT HOURS, RATE
?:N13
                 PRINT "NET EARNINGS = "RATE
?:M "RATE<"RATE
?:=
                 PRINT "NET EARNINGS = "RATE
?:M "RATE<",RATE
?:=
                 PRINT "NET EARNINGS = ",RATE
?:+1
                 GO TO START
?:M GO TO<GOTO
?:=
                 GOTO START
?:PP
```

Figure 3-3. File Editing (1 of 2)

```
?:PP

        NAME    DIM 15
        HOURS   FORM 2.2
        RATE    FORM 4.2
        TAX     FORM 4.2
        START   KEYIN *L,*L,*L
                KEYIN "PAY TO THE ORDER OF ", NAME
                KEYIN "HOURS WORKED ", HOURS
                KEYIN "RATE PER HOUR N", RATE
                KEYIN "TAX ", TAX
                MULT HOURS, RATE
                MULT RATE, TAX
                SUB TAX, RATE
                PRINT "NET EARNINGS = ",RATE
              >
                GOTO START
?:E

      ** FINAL OUTPUT ON UNIT 0, FILE EXAMPLE2  **
```

Figure 3-3. File Editing (2 of 2)

3.6 EDIT CALL USING EDIT PROMPTING

To call EDIT without supplying all parameters in the initial call statement, perform the following:

(1) Type in EDIT followed by a carriage return as follows:

```
EDIT↵
```

(2) Observe that the following message appears on the printer or CRT:

```
***EDITOR 1.1***
SOURCE FILE NAME (RETURN KEY IF NONE):
```

This message is significant only if an existing file is being edited.

(3) Perform a carriage return and observe that the following message appears on the printer or CRT:

```
SCRATCH FILE NAME:
```

This message is significant because it requests the name of the file to be filled.

(4) Enter the name of a file to be filled as follows:

```
SCRATCH FILE NAME:filename↵
```

NOTE

The filename specified in response to the above message does not have to be the name of an allocated file. If the filename is not the name of an allocated file EDIT allocates a file and assigns to it the filename specified and all remaining sectors on disk. The operator should truncate the file when completing the editing operation or with the FILES utility to free the unused disk space.

(5) Observe that the following message appears on the printer or CRT:

UNIT NUMBER:

(6) Enter the number of the logical unit containing the diskette where the file is located or is to be located (in the case where a nonexistent filename is given EDIT as a scratch file). Make the entry along with a carriage return as follows:

UNIT NUMBER =*n*↵

where *n* represents the selected logical unit number.

(7) Observe that the following message appears on the printer or CRT:

SELECT EDITOR LANGUAGE

"ASSEMBLER" EDITOR WILL BE ASSUMED, OTHERWISE TYPE (A) FOR "ASSEMBLER", (D) FOR "DACL-LAN" OR (T) FOR "TEXT":

(8) Enter the appropriate response followed by a carriage return as follows:

"...TEXT":*z*↵

where *z* is the appropriate response from the three possible responses, A, D, or T.

(9) Observe that the following message appears on the printer or CRT in response to the call statement:

*DO NOT REMOVE SOURCE MEDIA**
EDITOR 1.1
variable EDITOR READY

where *variable* is determined by what was input in the call statement for the parameter *z*: A produces ASSEMBLER; D, DACL LAN; and T, TEXT.

If the correct message does not appear in response to the call statement, abort the call by pressing the ESC key and enter the EDIT call once more.

If the correct message appears, EDIT is ready to accept data entered at the keyboard. However, before entering data into the file refer to paragraph 3.4 for information on operation of the software module EDIT.

SECTION 4 - COMPILING AND EXECUTING A DACL PROGRAM

4.1 INTRODUCTION

In the last section, the procedure for creating a file was described. This section describes how a file containing a DACL program is first compiled and, thereafter, executed on the System 3200. Two system software modules are involved in this process, one entitled DACL (the DACL compiler), which compiles the program, and the other called INT (interpreter), which executes the program. The manner in which each is called and the results produced is the subject of this section.

4.2 COMPILING A DACL PROGRAM

Two files are required by the DACL Compiler: a source file containing the DACL statements (an ASCII file) and a destination file to which the compiler writes the object code (a program file). If the destination file supplied to DACL has not been allocated by the operator prior to compilation, DACL allocates it during compilation providing there is room on the diskette. The destination file is truncated at end-of-data (EOD) automatically. (Take care when using a previously allocated file for the destination file as compilation will not complete if the file allocated is not large enough.)

4.2.1 Calling Sequence

To compile a DACL program, call DACL by typing the following:

[DACL][,*source file*][,*object file*]↵

Line Entry Item	Explanation
DACL	Command which calls DACL compiler into execution.

Line Entry Item	Explanation
source file	Filename of the file containing the DACL statements to be compiled.
object file	Name of the program file on the diskette to which the compiler is to write the object code. If the file is not already allocated, it will be allocated automatically and truncated to EOD after compilation. If it is already allocated, it will not be truncated after compilation.

If the operator types in "DACL" on the keyboard and does not identify the files, the following messages will appear. The operator must then type in the response indicated followed by a carriage return.

1. SOURCE FILE NAME:

   Type in name of file containing DACL statements.

2. UNIT NUMBER:

   Type in unit number of source file.

3. OBJECT FILE NAME:

   Type in name of program file in which object code is to be written.

4. UNIT NUMBER:

   Type in unit number of object file.

5. PRINT THE OUTPUT?

   Answer "YES" or "NO." "YES" will result in the instruction being printed.

6. DISPLAY THE OUTPUT?

   Answer "YES" or "NO." If "YES," the instructions will be displayed on the

CRT. (If the optional CRT is not part of the configuration, this will send output to the printer. Consequently, answering "YES" to both PRINT and DISPLAY will cause each instruction to be listed twice on the printer when there is no CRT.)

7. PRINT THE CODE?

Type "YES" or "NO." "YES" will cause the object code for each DACL statement to print in addition to the DACL statement.

After line entry is complete, the following message will appear.

PRINT THE HEADING:

Key in any identifying label to appear above the list of instructions. The heading may be left blank by depressing the RETURN key.

4.2.2 Example - Compiling a Program

In the following example, PAYROLL is the source file containing the DACL statements and PAY is the object file. The listing that results from the compilation of the DACL statements in the PAYROLL file is shown. Data typed by the operator is underlined.

```
:DACL PAYROLL/1.PAY/1
PRINT THE HEADING:
PAYROLL DEMONSTRATION
```

```
PAGE 01  PAYROLL DEMONSTRATION

 1.  3476  NAME     DIM 15
 2.  3488  HOURS    FORM 2.2
 3.  348F  RATE     FORM 4.2
 4.  3498  TAX      FORM 4.2
 5.  34A1  START    KEYIN *L.*L.*L
 6.  34A6           KEYIN "PAY TO THE ORDER OF ". NAME
 7.  34BD           KEYIN "HOURS WORKED ". HOURS
 8.  34CD           KEYIN "RATE PER HOUR N". RATE
 9.  34DF           KEYIN "TAX ". TAX
10.  34E6           MULT HOURS. RATE
11.  34E9           MULT RATE. TAX
12.  34EC           SUB TAX. RATE
13.  34EF           PRINT "NET EARNINGS = ",RATE
14.  3501           GOTO START
15.  3503           STOP
     3504           STOP

     3505  START

     3507  NAME
     3509  HOURS
     350B  RATE
     350D  TAX

 0 ERRORS...DONE
```

:

4.2.3 Diagnosis of Compiling Errors

To assist the programmer in debugging errors in his code, the compiler prints and displays error messages to the left of the listing. These error signals are an I, U, or E, with the following significance:

- I - Invalid or undefined instruction.
- U - Undefined label or variable.
- E - Error other than above - usually an error in order or arrangement.

In addition to providing error codes, each line is numbered to facilitate editing program errors. (See Editor, section 3.)

4.3 EXECUTION OF A PROGRAM (INTERPRETER)

When the programmer has successfully written and compiled an error-free program,

the object code (in the example, PAY) may be executed through the interpreter by:

1. Typing "INT" to call the interpreter program.

2. Typing the name of the object code file to be executed in response to the INT request for "PROGRAM NAME" which appears after typing "INT."

4.4 EXAMPLE - EXECUTION OF PROGRAM

On the following example, typing in "INT" and the object program name "PAY" causes execution of the instructions which were successfully compiled in the previous example. Underlined words indicate data entered by operator at the keyboard.

```
:INT
/D A C L  1.1
PROGRAM NAME: PAY

PAY TO THE ORDER OF EMPLOYEE 1
HOURS WORKED 42.00
RATE PER HOUR N 5.39
TAX  .20
NET EARNINGS =  181.10

PAY TO THE ORDER OF EMPLOYEE 2
HOURS WORKED 49.00
RATE PER HOUR N 4.32
TAX  .19
NET EARNINGS =  171.46

PAY TO THE ORDER OF EMPLOYEE 3
HOURS WORKED 37.00
RATE PER HOUR N 3.39
TAX  .18
NET EARNINGS =  102.85
```

SECTION 5 - ASSEMBLING AND EXECUTING AN ASSEMBLY LANGUAGE PROGRAM

5.1 INTRODUCTION

This section describes how a file, created using software module EDIT, as described in Section 3, can be assembled and, thereafter, executed on the System 3200. The system software module ASEM is involved in this process. The manner in which ASEM is called and the output produced by ASEM is the subject of part of this section. The remainder of the section is devoted to describing the system program XREF which prints a cross-reference listing and to describing what is involved in executing an assembled program.

5.2 ASSEMBLING A PROGRAM

ASEM assembles the source statements of an assembly language program, contained in an ASCII file, into object code which it then places in a separate program file. (Program file refers to the file type assigned to a file when it is first allocated.) In addition, ASEM also produces a cross-reference file of labels used in the assembly language program. ASEM is a multi-module assembler, i.e., it assembles at one time a collection of modules each separated by an END statement and each containing a program or collection of assembly language statements. In an assembly containing several modules, labels that are common between modules either contain a colon (:) or begin with an @ character, e.g., @TAG and TAG:1. These labels are called global labels. The labels of memory resident system subroutines which may be called by any assembly language program are global labels. A directory of these labels is contained in a file entitled TEXT:*xx**, a part of the system software. TEXT:*xx** associates with global label with its memory address. Thus if an assembly language program uses a system subroutine, TEXT:*xx** must also be included in the assembly of the program uses a that the label is translated into a meaningful memory address. This is accomplished by combining TEXT:*xx** with the file containing the assembly language program into still another file which can then be assembled.

*The *xx* portion of this filename differs depending on the level of the software present in a system. To determine what the numbers for a system are, perform the LST command of the FILES utility on the system deskette (see Section 6).

5.3 CALLING ASEM

ASEM is called and an assembly executed by typing the following:

[ASEM][,*source filename*](,*object filename*)(,*list option*)(,*cross-reference filename*)↵

Line Entry Item	Description
ASEM	Command which calls ASEM into execution.
source filename	The name of the file containing the assembly language source statements.
object filename	The name of the file to be used to contain the assembled object code.
list option	Three possible options exist: L, N, or a number between 0-127. L = list entire assembly on the printer. N = turn entire listing off except for errors. n = 0-127 and causes ASEM to skip listing of the first n modules. List statements contained in the source module will be in effect. Default for the list option is 0. With 0 selected intentionally or by default, the listing can be manually turned on or off by typing an L or N on the console during the execution of ASEM.
cross-reference filename	The name of file to contain the cross-reference file generated by ASEM during the assembly process.

The command ASEM and all filenames follow the convention of containing an optional unit designator, i.e., ASEM/2,.... would load the assembler from logical unit 2. Otherwise, all files are assumed to reside on logical unit 0.

EXAMPLES

1. ASEM,SRC ↵

   File SRC on unit 0 is assembled and listed. No object or cross-reference files are created.

2. ASEM,SRC,SRCBIN/1,N ↵

   File SRC is assembled with no listing. Object file SRCBIN is written on unit 1.

3. ASEM/1,SRC/2,,,SRCXREF ↵

   The assembler is loaded from unit 1. File SRC on unit 2 is assembled and listed. No object file is created but cross-reference file SRCXREF is written on unit 0.

4. COPY TEXT:*xx**,MOD1, MOD2,SRC ↵
   ASEM,SRC,SRCBIN,2,SRCXREF ↵

   (See Section 6 for a complete description of COPY.

   Modules TEXT:*xx**,MOD1 and MOD2 are combined into one source file SRC. SRC is assembled, creating object file SRCBIN and cross-reference file SRCXREF. The listing begins with the third module (MOD2).

5.4 XREF EXECUTION

The program XREF prints the cross-reference listing from the cross-reference file created during an assembly. XREF is executed by typing the following command:

[XREF,][,*cross-reference filename*](,*list option*) ↵

*The *xx* portion of this filename differs depending on the level of the software present in a system. To determine what the numbers for a system are, perform the LST command of the FILES utility on the system diskette (see Section 6).

Line Entry Item	Description
cross-reference filename	Name of a cross-reference file created by ASEM during assembly of a program.
list option n	*n* = a number between 0-127. No cross-reference listing is produced for the first n modules.

If no list option is entered, the first module will be skipped and no global cross-reference will be produced. This would be the normal mode if file TEXT:*xx** is used as the first module and a lengthy global cross-reference is not desired.

There must be at least 260 sectors available for a scratch file on the same diskette with the cross-reference file. If not, the file must be copied to another diskette before executing XREF. The scratch file is automatically allocated and deleted by XREF.

EXAMPLES

1. XREF,SRCXREF,0 ↵

   Cross-reference file SRCXREF is listed. A global cross-reference is also produced.

2. XREF,SRCXREF/1 ↵

   Cross-reference file SRCXREF on unit 1 is listed. The first module is skipped and no global cross-reference is produced.

*The *xx* portion of this filename differs depending on the level of the software present in a system. To determine what the numbers for a system are, perform the LST command of the FILES utility on the system diskette (see Section 6).

SECTION 6 - SYSTEM UTILITY SOFTWARE

6.1 INTRODUCTION

The system utilities provide the operator essential capability to (1) handle files, (2) handle diskettes, (3) perform diagnostics, and (4) to display and set the system clock. A utility entitled FILES, which comprises twelve separate functions, provides the operator file handling capability. A collection of seven utilities serves the operator with diskette handling capability. Four utilities supply the operator with diagnostic tests and idskette repair capability. Finally, two utilities entitled TIME and TIME SET permits the operator to display and set the system clock.

Throughout this section, all programs can be called using the *program name/unit no.* convention, where the optional */unit no.* suffix to a program name (filename) indicates the logical unit number of the drive containing the diskette where the program (file) is stored.

6.2 THE FILES UTILITY

FILES provides the operator the ability to examine and change the file labels in tracks 0 and 1, the diskette file directory, of the specified diskette. See Table 2-3 for a description of a file label. In addition, FILES permits the operator to write the volume label (see the System Programmer Reference Manual for a description of a volume label) for a specified diskette. Also, FILES permits the operator to display the file label(s) in the file directory for a single file, a group of files, or all files on a diskette. Finally, FILES allows the operator to change FILES operation from a diskette in one logical unit to another diskette in another logical unit. Each function is selected by typing the appropriate command from the following set: LST, FLS, ALO, DEL, TRU, REN, PRO, UNP, TYP, LBL, DAT, and UNT. Each of these command keywords perform one of the functions described in the following paragraphs. These commands are specified by the operator as a line entry after the system utility, FILES, has been called into execution in the system. The format of the keyboard line entry which calls FILES into execution is as follows.

[FILES(/*unit no.*)(,*unit no.*)↵

Line Entry Item	Explanation
FILES(/*unit no.*)	The keyword FILES causes Command Processor to load and execute the diskette file labeled FILES.
unit	A decimal digit which specifies the logical unit containing the diskette directory file labels which FILES will access. If no value is specified, the default condition specifies logical unit 0.

FILES provides a message containing the various commands available. The message is output when the operator calls FILES into execution. Some of the individual command keywords also display a message requesting required parameters. An example of the FILES keyboard line entry is shown below.

```
FILES/0,1↵

ENTER COMMAND (LST,FLS,ALO,DEL,TRU,REN,PRO,UNP,TYP,LBL,DAT,UNT)
```

The utility FILES on the diskette in logical unit 0 is called to operate on the directory file labels of the diskette in logical unit 1. FILES responds by displaying the ENTER COMMAND... message.

Pressing the ESCape key permits the operator to exit FILES operation and return to operation with the command processor.

6.2.1 List-File (LST) Command

The list-file option displays the contents of the file label(s) contained in the diskette file directory for an individual file, a specific group of files or all files on the diskette. The command parameter specifies which of these three possible operations are performed. The format of the keyboard line entry for the list-file command is shown below.

[LST](,*filename* or *variable**)↵

Line Entry Item	Explanation
LST	The keyword LST informs FILES that the list-file command is to be be executed.
filename or *variable**	If this optional line entry item is omitted, the contents of all file labels in the diskette directory are printed. If a single filename is specified, the file label contents for that file in the directory is printed. Specifying a single file precludes the use of the *variable**. *Variable** permits the operator to specify an ordered set of alphanumeric characters followed by an asterisk (*). A maximum of seven characters can be specified. Specifying the alphanumeric characters followed by an * causes the file labels of all files in the directory whose filenames begin with the letters specified by the ordered set of alphanumeric characters to be printed.

An example of the keyboard line entry for the list-file command and the response displayed by FILES is shown below.

LST,FILE*↵

The file labels in the diskette file directory for all files on the diskette whose filenames begin with the word FILE are displayed by FILES.

```
FILES ON UNIT 0  VOL VOL1                                    J.PROG

  NAME    TYP RF RLN  BOE    EOE   SECS  FP MV SEQ  CREATED    EOD   UNUSED SECS

FILES      B     000 04 21 06 05    37   P          05/28/76  06 06         0
FILE1      A     000 64 23 64 23     1              07/09/76  64 24         0
FILE2      A     000 64 24 64 24     1                / /     64 25         0
FILE3      A     000 64 25 64 25     1                / /     64 25         1
FILE4      A     000 64 26 65 24    25                / /     65 01        24
FILE5      A     000 65 25 66 23    25                / /     65 26        24

UNUSED FILE NAMES  06

UNUSED SECTORS
 BOE   NUMBER
31 01    850
66 25     48
70 04    179
ENTER COMMAND
```

6.2.2 List-Filename (FLS) Command

The list-filename option displays the filename(s) and file type(s) in the file-label(s) contained in the diskette directory for an individual file, a specific group of files, or all files on a diskette. The command parameter specifies which of these three possible operations are performed. The format of the keyboard line entry for the list-filename command is shown below.

[FLS](,*filename* or *variable**) ↵

Line Entry Item	Explanation
FLS	The keyword FLS informs FILES that the list-filename command is to be executed.
filename or *variable**	If this optional line entry item is omitted, the filename and file type in the file labels contained in the diskette directory for all files on diskette are printed. If a single filename is specified, the filename and file type in the directory file label for that file is printed. Specifying a single file precludes the use of the *variable**. *Variable** permits the operator to specify an ordered set of alphanumeric characters followed by an asterisk (*). A maximum of seven characters can be specified. Specifying the alphanumeric characters followed by an * causes the filename(s) and file type(s) in the file labels of all files whose filenames begin with the letters specified by the ordered set of alphanumeric characters to be printed.

An example of the keyboard line entry for the list-filename command and the response displayed by FILES is shown below.

FLS,FILE* ↵

The filename and file type in the diskette file directory file label for all files on the diskette whose names begin with word FILE are displayed by FILES.

```
FILES ON UNIT 0   VOL VOL1                                    J.PROG

  NAME    TYPE

FILES       B
FILE1       A
FILE2       A
FILE3       A
FILE4       A
FILE5       A

UNUSED FILE NAMES  06

UNUSED SECTORS
 BOE    NUMBER
31 01     850
66 25      48
70 04     179
ENTER COMMAND
```

6.2.3 Allocate-File (ALO) Command

The allocate-file command allocates the amount of diskette storage space requested by the operator for the new file specified by the command. FILES queries the operator by displaying messages immediately after the operator enters the allocate-file keyboard line entry. The format of the keyboard line entry for the allocate-file command is shown below.

[ALO](,*filename*)↵

Line Entry Item	Explanation
ALO	The keyword ALO informs FILES that the allocate-file command is to be executed.
filename	The name of the file to be allocated.

An example of the keyboard line entry for the allocate-file command is shown below.

ALO,FILE↵

Immediately after the operator makes the keyboard line entry, FILES prints the following message.

```
SECTORS? (NO ENTRY=MAX AVAILABLE)3↵
TYPE? (A,B,P,S)A↵
RECORD FORMAT? (V,SPACE)↵
RECORD LENGTH? (1-255)↵
```

The question, SECTOR?, asks how many sectors are to be allocated to the file. If the operator does not specify an amount, all available sectors on diskette are allocated. TYPE? asks what file type to assign to the file. Similarly, the question, RECORD FORMAT?, asks what types of records the file will contain. RECORD LENGTH asks for the size of the records (in bytes) to be used in the file. The default condition specifies 0 bytes. RECORD LENGTH is not displayed if V (variable) RECORD FORMAT is specified. The answer to these questions are placed in the file label of the file

being allocated. Thereafter, FILES displays the following message to indicate successful completion of its operation.

FILE FILE ALLOCATE ON UNIT *n* VOL *volume id*

6.2.4 Delete-File (DEL) Command

The delete-file command deletes the file specified by the command from the diskette. The format of the keyboard line entry for the delete-file command is shown below.

[DEL][,*filename*]↵

Line Entry Item	Explanation
DEL	They keyword DEL informs FILES that the delete-file command is to be executed.
filename	The name of the file to be deleted.

An example of the keyboard line entry for the delete-file command is shown below.

DEL,FILE1↵

If the file is not write-protected, FILES deletes the file and displays the following message.

FILE FILE1 DELETED ON UNIT *n* VOL *volume id*

If the file specified is write-protected, a message similar to the following is displayed. The message informs the operator that the file is protected, thus the operator must perform an unprotect-file command on the file in order to delete it.

FILE *filename* WRITE PROTECTED ON UNIT *n* VOL *volume id*

6.2.5 Truncate-File (TRU) Command

The truncate-file command truncates any unused space from the end of the file specified. The format of the keyboard line entry for the truncate-file command is shown below.

[TRU][,*filename*]↵

Line Entry Item	Explanation
TRU	The keyword TRU informs FILES that the truncate-file command is to be executed.
filename	The name of the file to be truncated.

An example of the keyboard line entry for the truncate-file command is shown below.

TRU,FILE2 ↵

If the file is not write-protected, FILES truncates the file and displays the following message.

FILE FILE2 TRUNCATED ON UNIT *n* VOL *volume id*

If the file specified is write-protected, a message similar to the following is displayed. The message informs the operator that the file is protected, thus the operator must perform an unprotect-file command on the file in order to truncate it.

FILE *filename* WRITE PROTECTED ON UNIT *n* VOL *volume id*

6.2.6 Rename-File (REN) Command

The rename-file command assigns a new name to the file specified. The format of the keyboard line entry for the rename-file command is shown below.

[REN][,*filename*] ↵

Line Entry Item	Explanation
REN	The keyword REN informs FILES that the rename-file command is to be executed.
filename	The name of the file to be renamed.

An example of the keyboard line entry for the rename-file command is shown below.

REN,FILE2 ↵

Immediately after the rename-file command is entered on the keyboard, the following message prints which requests the new name to be assigned to the file. The operator responds by entering the new filename to be assigned to the file. In the example the new filename to be assigned to FILE2 is NEWFILE.

NEW NAME?NEWFILE ↵

FILES displays the following message to indicate successful completion of its operation.

FILE FILE2 RENAMED NEWFILE ON UNIT *n* VOL *volume id*

6.2.7 Write-Protect (PRO) Command

The write-protect command assigns the write-protect attribute to the file specified. This prevents the file from being inadvertently written over, truncated, or deleted. The format of the keyboard line entry for the write-protect command is shown below.

[PRO][,*filename*] ↵

Line Entry Item	Explanation
PRO	The keyword PRO informs FILES that the write-protect command is to be executed.
filename	The name of the file to receive the write-protect attribute.

An example of the keyboard line entry for the write-protect command and the response displayed by FILES is shown below.

PRO,FILE1 ↵

FILE FILE1 WRITE PROTECTED ON UNIT *n* VOL *volume id*

6.2.8 Unprotect-File (UNP) Command

The unprotect-file command removes the write-protect attribute from the file specified. This permits the file to be written over, truncated, or deleted. The format of the keyboard line entry for the unprotect-file command is shown below.

[UNP][,*filename*]↵

Line Entry Item	Explanation
UNP	The keyword UNP informs FILES that the unprotect-file command is to be executed.
filename	The name of the file from which the write-protect attribute is to be removed.

An example of the keyboard line entry for the unprotect-file command and the response displayed by FILES is shown below.

```
UNP,FILE1↵
FILE FILE1          UNPROTECTED ON UNIT n     VOL volume id
```

6.2.9 Retype-File (TYP) Command

The retype-file command assigns a new file type attribute to the file specified. Thus where before a file was typed as an ASCII file, it may be retyped to a binary file, etc. The format of the keyboard line entry for the retype-file command is shown below.

[TYP][,*filename*]↵

Line Entry Item	Explanation
TYP	The keyword TYP informs FILES that the retype-file command is to be executed.
filename	The name of the file whose file type is to be changed.

An example of the keyboard line entry for the retype-file command is shown below.

TYP,FILE1 ↵

Immediately after the retype-file command is entered on the keyboard, the following message is displayed which requests the new file type to be assigned to the file. The operator responds by entering the new file type to be assigned to the file: A (ASCII), B (Binary), P (Program Load File), and S (System Load File). In the example the operator responds by specifying an ASCII (A) file type.

NEW TYPE? A ↵

Thereafter, FILES displays the following message to indicate successful completion of its operation.

FILE FILE1 ASSIGNED TYPE A ON UNIT *n* VOL *volume id*

6.2.10 Label (LBL) Command

The label command writes a new volume label for the diskette contained in the diskette logical unit specified in the initial keyboard line entry that called FILES into execution. See the System Programmer Reference Manual for a description of volume label. The format of the keyboard line entry for the label command is shown below.

[LBL] ↵

Line Entry Item	Explanation
LBL	The keyword LBL informs FILES that the label command is to be executed.

An example of the keyboard line entry for the label command is shown below.

LBL ↵

Immediately after the label command is entered on the keyboard, the following message prints which requests the new parameters to be placed in the volume label. The operators respond by entering each parameter requested.

```
NEW VOL ID? VOL2
NEW OWNER ID? OWNER'S NAME
```

Thereafter, FILES displays the following message to indicate successful completion of its operation.

```
NEW VOL ID VOL2          OWNER'S NAME          WRITTEN ON UNIT n
```

6.2.11 Date-File (DAT) Command

The date-file command inserts a date in the diskette directory file label of the file specified. The format of the keyboard line entry for the label is shown below.

[DAT][,*filename*]

Line Entry Item	Explanation
DAT	The keyboard DAT informs FILES that the date-file command is to be executed.
filename	The name of the file which will be assigned a date.

An example of the keyboard line entry for the date-file command is shown below.

```
DAT FILE1
```

Immediately after the date-file command is entered on the keyboard, the following message prints which requests the date to be placed in the file label for the file. The operator responds by entering the appropriate date.

```
ENTER DATE    YYMMDD    760704
```

Thereafter, FILES displays the following message to indicate successful completion of its operation.

FILE FILE1 DATED ON UNIT *n* VOL *volume id*

6.2.12 Change-Unit (UNT) Command

In the keyboard line entry which calls FILES into execution, one of the unit designators which may optionally appear after the keyword FILES (see paragraph 6.2) specifies the logical unit containing the diskette directory file labels which FILES will access. If not specified, the default condition specifies logical unit 0. The change-unit command permits the operator to change this unit designation from one logical unit to another. The format of the keyboard line entry for the change-unit command is shown below.

[UNT]↵

Line Entry Item	Explanation
UNT	The keyword UNT informs FILES that the change-unit command is to be executed.

Immediately after the operator makes the keyboard line entry, FILES prints the following message which requests the new logical unit number. The operator responds by entering the new unit number, 2.

UNT?2↵

6.3 THE DISKETTE HANDLING UTILITIES

The diskette utilities are a collection of routines which permit the operator to structure blank diskettes in the System 3200 diskette sector organization and to transfer files from one diskette to another or from diskette to either the CRT or printer. Seven routines perform these functions: FORMAT (format-diskette), INIT (initialize-diskette), DUMP (dump-file), PRINT (print-file), COPY (copy-file), SAVE (save-file), and DISKCOPY (copy-diskette).

6.3.1 FORMAT (Format-Diskette)

FORMAT allows the operator to format a blank diskette. This formatted diskette can then be initialized by the INIT routine and thereafter used in the system. Formatting a blank diskette involves writing the track and sector ID records of a specified number of tracks on a diskette. Refer to the System Programmer Reference Manual for more information on the diskette file structure. The line entry items accompanying the FORMAT keyword specifies the unit containing the diskette to be formatted; which tracks on the diskette are to be formatted, i.e., any group of tracks from track 0 through track 76; the pattern of data to be stored in the sectors being formatted; and the interlacing scheme for assigning addresses to sectors. To further explain what is meant by interlacing scheme, consider a track containing 26 sectors. The operator assigns a value between 1 and 25 as the interlace value in the FORMAT line entry, for example, 4. The value 4 specifies that once the ID record for sector 1 has been written, the fourth sector following sector 1 is formatted as sector 2, the fourth sector from sector 2 as sector 3, etc. A value of 5 specifies that the fifth sector from sector 1 is formatted as sector 2, etc. A track formatted with an interlace value of 4 and another with a value of 5 is shown in the following illustration. The FORMAT software assigns the sectors in several passes and thereafter writes the sectors physically on the diskette. Note that the interlace value of 5 produces a symmetrical arrangement of sectors; whereas, the value 4 does not. The symmetry ends for the interlace value 4 after the second pass. Sector 13 is assigned to the 23rd sector on the diskette, then FORMAT attempts to assign sector 14 to the fourth sector from the 23rd sector but finds it already assigned as sector 1. It then places sector 14 in the next available sector after sector 1.

Interlace Value 4

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
1	14	8	21	2	15	9	22	3	16	10	23	4	17	11	24	5	18	12	25	6	19	13	26	7	20

1				2				3				4				5				6				7		PASS 1
		8				9				10				11				12				13				PASS 2
	14				15				16				17				18				19				20	PASS 3
			21				22				23				24				25				26			PASS 4

Interlace Value 5

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
1	22	17	12	7	2	23	18	13	8	3	24	19	14	9	4	25	20	15	10	5	26	21	16	11	6

1					2					3					4					5					6	PASS 1
				7					8					9					10					11		PASS 2
			12					13					14					15				16				PASS 3
		17					18					19					20				21					PASS 4
	22					23					24					25					26					PASS 5

In the diskette file structure, each sector contains a sector ID record which comprises of one byte containing sector address and another containing track address. All sectors of one track are unique from sectors of another track because of track address. FORMAT writes both track and sector addresses.

The format of the keyboard line entry for FORMAT is shown below.

[FORMAT(/*unit no.*)][,*unit no.*][,*from track*][,*to track*][,*pattern*][,*interlace value*]↵

FORMAT also provides the operator a prompting message in response to the operator entering only the keyword FORMAT followed by a carriage return. This is shown in the following example.

FORMAT/1 ↵

ENTER UNIT,FROM TRACK,TO TRACK,PATTERN(R,I,HEX),OPT(1-25)

Line Entry Item	Explanation
FORMAT/*unit no.*	The keyword FORMAT initiates the program FORMAT.
unit no.	A number other than 0 which indicates the logical unit containing the diskette to be formatted.
from track	A number from 0 through 76 which specifies the track at which formatting will begin. This number must be smaller than the number specified as the *to track* line entry item.
to track	A number from 0 through 76 which specifies the track at which formatting will conclude. This number must be the same or larger than the number specified as the *from track* line entry item.
pattern	Three patterns of data can be specified represented by the mnemonics listed below.

Mnemonic	Pattern
I	Pattern I specifies that a byte of data formulated by multiplying the track address t of a sector which is to be filled by the constant 32 and adding the product to the sector address s. Thereafter this sum is ANDed with the hexadecimal value FF: $[(t * 32) + S]$ AND $FF_{(16)} = I$. This logical product is then stored in the first byte of a sector. For each subsequent byte in the sector, the product is incremented by +1. Thereafter, a calculation is made for the next sector and the process continues until all sectors are filled.
R	Pattern R is a random pattern created by generating and filling each sector with a unique random number.

Line Entry Item	Explanation	
	Mnemonic	Pattern
pattern (cont.)	$xx_{(16)}$	The variable $xx_{(16)}$ is meant to imply two hexadecimal digits. These digits are the 8-bit value stored in each byte of every sector.
interlace value	The interlace value is any number from 1 through 25. An explanation of the purpose of the value is contained in the description of the Format-Diskette program.	

Once the keyboard line entry is made, FORMAT executes up to a point then prints the following message:

DISK FORMATTER, ENTER Y TO CONTINUE

To continue, the operator must type an upper case Y after which FORMAT completes by typing: ALL DONE.

6.3.2 INIT (Initialize-Diskette)

INIT initializes tracks 0 and 1 of any diskette mounted on a diskette drive other than logical unit 0. INIT stores the entry DDR1 in the LABEL ID field of sectors 8 through 26 of the diskette track 0 and 1 through 26 of track 1. Sectors 8 through 26 of track 0 and 1 through 26 of track 1 form the diskette file directory. Each sector contains a file label which describes each file on the diskette. The LABEL ID field of each file label contains an HDR1 or DDR1 entry to indicate that the file described by the file label is either an active or a null file, respectively. By inserting DDR1 in sectors 8 through 26 of track 0 and 1 through 26 of track 1, INIT creates an empty diskette. See the diskette file structure description in the System Programmer Reference Manual for a discussion of the diskette file structure. The format of the keyboard line entry for INIT is shown below.

[INIT(/*unit no.*)](,*unit no.*)↵

INIT also provides the operator a prompting message in response to the operator entering only the keyword INIT followed by a carriage return. This is shown in the following example.

INIT ↵

ENTER UNIT NO.

Line Entry Item	Explanation
INIT(/*unit no.*)	The keyword INIT initiates the program INIT.
unit no.	A number other than 0 which indicates the logical unit containing the diskette to be initialized.

Once the keyboard line entry is made, INIT executes up to a point then prints the following message:

DISK INITIALIZER, ENTER Y TO CONTINUE

To continue, the operator must type an upper case Y after which INIT completes by printing: ALL DONE.

6.3.3 DUMP (Dump-File)

DUMP displays the file specified in the keyboard line entry on the printer or CRT. The contents of the file are represented in hexadecimal and compressed ASCII in the printout or on the CRT screen. Compressed ASCII is ASCII characters with blank suppression. In addition, the track number and sector number are likewise displayed.

There are two formats for the keyboard line entry for DUMP. The first contains the keyword DUMP followed by a filename. The second contains the keyword DUMP followed by the track and sector number of the location on diskette where the dump will begin followed by the track and sector number of the location on diskette where the dump will end. Each of these is listed below.

First Format:

[DUMP(/*unit no.*)][,*filename*]↵

Second Format:

[DUMP(/*unit no.*)][,*unit*][,*from-trk/sec*][,*to-trk/sec*]↵

DUMP also provides the operator a prompting message in response to the operator entering only the keyword DUMP followed by a carriage return. This is shown in the following example.

DUMP↵

ENTER: UNIT,FROM-TRK/SEC,TO-TRK/SEC OR FILENAME

Line Entry Item	Explanation
DUMP(/*unit no.*)	The keyword DUMP initiates the program DUMP.
filename	The name of the file on diskette that is to be dumped.
unit no.	Specifies the logical unit containing the diskette from which data is to be dumped.
from-trk/sec	The track number followed by a comma (,) delimiter followed by a sector number of the location on diskette at which the dump will begin.
to-trk/sec	The track number followed by a comma (,) delimiter followed by a sector number of the location on the diskette at which the dump will end.

Immediately after the operator enters a keyboard line entry for DUMP, the requested file or requested track(s) and sector(s) are displayed. Below is shown samples of the two types of keyboard line entries followed by a sample printout produced by DUMP.

Keyboard Line Entry First Format:

```
DUMP,AFILE
```

Keyboard Line Entry Second Format:

```
DUMP,1,25,3,25,4
```

Sample DUMP Printout:

```
TRACK  25   SECTOR   3

12 03   45 51   55 12   03 31   32 12   0D 44   3D 45   4E 44   *..EQU..12..D=END*
49 4E   47 20   53 45   43 54   4F 52   2C 20   45 3D   45 4E   *ING SECTOR, E=EN*
44 49   4E 47   20 54   52 41   43 4B   2E 0D   46 3A   52 53   *DING TRACK..F:RS*
12 04   45 51   55 12   03 31   34 12   0D 44   45 3D   4E 45   *..EQU..14..DE=NE*
58 54   20 52   45 4C   41 54   49 56   45 20   53 45   43 54   *XT RELATIVE SECT*
4F 52   20 4E   42 52   2E 0D   12 07   50 47   45 0D   12 07   *OR NBR....PGE...*
45 4E   44 0D   12 07   4C 49   53 54   20 20   31 0D   12 08   *END...LIST  1...*
4F 52   47 12   03 55   3A 42   45 47   0D 0D   50 52   49 4E   *ORG..U:BEG..PRIN*


TRACK  25   SECTOR   4

54 12   03 4D   56 49   12 03   41 2C   23 44   0D 12   08 43   *T..MVI..A,#D...C*
41 4C   4C 20   20 53   3A 50   52 54   0D 12   08 4D   56 49   *ALL  S:PRT...MVI*
12 03   41 2C   23 41   0D 12   08 43   41 4C   4C 20   20 53   *..A,#A...CALL  S*
3A 50   52 54   0D 12   08 4D   56 49   12 03   45 2C   27 30   *:PRT...MVI..E,'0*
27 0D   12 08   4D 56   49 12   03 43   2C 31   33 30   0D 0D   *'...MVI..C,130..*
50 4C   4E 4C   12 04   4D 4F   56 12   03 41   2C 45   0D 12   *PLNL..MOV..A,E..*
08 43   41 4C   4C 20   20 53   3A 50   52 54   0D 12   08 4D   *.CALL  S:PRT...M*
4F 56   12 03   42 2C   43 0D   12 08   4D 56   49 12   03 41   *OV..B,C...MVI..A*


ALL DONE

:
```

6.3.4 PRINT (Print-File)

PRINT displays the file specified in the keyboard line entry on the printer or CRT. The contents of the file are represented in ASCII without compression. This is in contrast to the dump-file utility which displays the file with compression. One additional feature of the PRINT utility on CRT systems is the ability to print the file contents and not the keyboard line entry that calls the PRINT utility. This is accomplished by setting switch 0 on prior to making the PRINT keyboard line entry. Thereafter, the keyboard line entry will appear on the CRT and the file contents will appear on the printer. The format of the keyboard line entry for PRINT is shown below.

[PRINT(/*unit no.*)][,*filename*]↵

Line Entry Item	Explanation
PRINT(/*unit no.*	The keyword PRINT initiates the program PRINT.
filename	The name of the file on diskette that is to be printed.

A sample of PRINT operation is shown in the following example.

PRINT,FILE1↵

```
        ORG   #2000

START   XRA   A
        STA   #999
        RET

        END   START


ALL DONE
:
```

6.3.5 COPY (Copy-File)

COPY copies or appends the contents of a file or files specified as source into or onto a file specified as destination. The source and destination files are input together in the keyboard line entry separated by commas. When more than one filename is specified, the last filename is assumed to be the destination file. The destination file must already be allocated. (See the allocate-file command of system utility, FILES, for file allocation.) The format of the keyboard line entry for COPY is shown below. If specifying options append, protect, and/or close, the letters A, P, or C must procede the option specified. In addition, the options may be specified in any order.

[COPY][,*source 1*(/*unit no.*),*source 2*(/*unit no.*),...*source n*(/*unit no*)
destination(/*unit no.*)](,A=*append*)(,P=*protect*)(,C=*close*)↵

COPY also provides the operator a prompting message in response to the operator entering only the keyword COPY followed by a carriage return. This is shown in the following example.

COPY
↵

ENTER: SOURCE FILENAME(S),DEST FILENAME(,OPT)

OPT ARE:

C=0(CLOSE AT EOD) C=1(CLOSE AT EOE)
C=2(TRUNK AT EOD)
P=1(PROTECT) A=1(APPEND)

ENTER COMMAND

Line Entry Item	Explanation
COPY(/*unit no.*)	The keyword COPY initiates the program COPY.
source 1(/*unit no.*), *source 2*(/*unit no.*),... *source n*(/*unit no.*)	The name of the file or files to be used as the source in the transfer.

Line Entry Item	Explanation
destination/unit no.	The name of the file to be used as the destination in the transfer. The file must exist on the destination diskette, not be write-protected, and have sufficient space allocated to contain the files being copied. If the append option is specified, the destination file initial contents are preserved; otherwise, the initial contents are overwritten.
A=*append*	The line entry item can take the value 0 or 1. If the value 0 is specified, COPY overlays the contents of the destination file. If a 1 is specified, COPY appends the source file to the destination file from end-of-data (EOD) to end-of-extent (EOE). EOD is the last location in the file containing data. EOE is the last location allocated to the file. In order for an append operation to occur, EOE must exceed EOD by the amount of space required to contain the source file. If not specified the source file overlays the destination file.
P=*protect option*	This line entry item specifies the type of file protection to be given the destination file. The type of protection can be specified by entering 0 or 1 as the *protect option* or by omitting the line entry item altogether. Specifying a 0 directs COPY to leave destination file unprotected, while specifying a 1 directs COPY to write-protect the file.
C=*close option*	The line entry item specifies how the file copied on the destination diskette is to be closed. The *close option* takes the form of a number from 0 through 3. The numbers specify the following close options. If the line entry is omitted, the default condition is C=0.

Line Entry Item	Explanation	
C=*close option* (Continued)	*close option*	Meaning
	0	Close file at EOD (end-of-data)
	1	Close file at EOE (end-of-extent)
	2	Close file at EOD and truncate file to EOD.

Default conditions exists for each line entry item omitted by the operator. The default for T=*file type* is all file types; for the C=*close option*, close file at EOD (C=0); and for the P=*protect option*, do not write-protect (P=0).

COPY indicates successful completion of its operation by printing the message:

FILES COPIED TO FILE *filename* ON UNIT *unit number* ARE

filename 1 *source unit no.*
filename 2 *source unit no.*
.
.
.
filename n *source unit no.* ALL DONE

A sample of COPY operation is shown in the following example.

```
COPY,SFILE1,SFILE2,DFILE/1,P=0,A=0,C=0↵

FILES COPIED TO FILE DFILE          ON UNIT 1 ARE

SFILE1    0
SFILE2    0     ALL DONE
```

In the example, the third file, DFILE, is interpreted by COPY to be the destination file. Note that the /1 designation on DFILE informs COPY that the destination file is on unit 1. If not otherwise specified, COPY assumes that all files are on unit 0.

6.3.6 SAVE (Save-File)

SAVE allows the operator to move selected files from one diskette (source) to another diskette (destination). The keyboard line entry input permits the operator to save all files from the source diskette on the destination diskette, to save only files of a specific file type, or to save files by filename. SAVE allocates space for the files being saved on the destination diskette. (See the allocate-file command of system utility, FILES, for an explanation of file allocation.) The format of the keyboard line entry for SAVE is shown below. See the diskette file structure description in the System Programmer Reference Manual for more information on the terms used in this paragraph (e.g., file type, EOD, EOE, etc.).

```
[SAVE(/unit no.)][,filename 1,...filename n, and/or filename*, and/or *filename, and/or
    *filename*, and/or *x*x*...*x*)(,T=file type)(,P=protect option)(,C=close option)
(,S=source unit)(,D=destination unit)↵
```

SAVE also provides the operator a prompting message in response to the operator entering only the keyword SAVE followed by a carriage return. This is shown in the following example.

SAVE

ENTER FILENAME(S)(*) (,OPT)

OPT ARE:

S=SOURCE D=DEST T=TYPE(S)
C=0(CLOSE AT EOD) C=1(CLOSE AT EOE)
C=2(TRUNK AT EOD) C=4(ALO ALL CLOSE AT EOD)
C=5(ALO ALL CLOSE AT EOE)

ENTER COMMAND

Line Entry Item	Explanation
SAVE(/*unit*)	The keyword SAVE initiates the program SAVE.
filename 1,...filename n *filename** **filename* **filename** **x*x*...*x** *	This optional line entry item permits the operator to specify to be saved one or more filename and/or an asterisk (*) followed by all or part of a filename, and/or all or part of a filename followed by an asterisk, and/or all or part of a filename preceded and followed by an asterisk. An asterisk before a filename causes SAVE to save all files having the letters following the asterisk at the end of their filename. An asterisk following a filename causes SAVE to save all files having the letters preceding the asterisk at the start of their filename. An asterisk on either side of a filename causes SAVE to save all files having the letters enclosed by the asterisks in the middle of their filename. In addition, the line entry item also permits the operator to specify to be saved all filenames which have the sequence of letters *x*x*...*x* specified in the line entry. As shown, the sequence set off by asterisks, F*I*L*E*, will save all files whose filenames contain the order sequence of letters "FILE" anywhere in the name. A maximum of eight characters can be specified. Specifying * alone saves all file from unit 0 to unit 1.

Line Entry Item	Explanation
T=*file type(s)*	Four types of files can be stored on a system diskette and all files on a diskette must be classified as one of these four types. Thus the SAVE line entry permits the operator to specify all files of a given type(s) on the source diskette are to be saved. The file types are specified by letter mnemonics following the "T=" position of the line entry. The "T=" portion of the line entry must precede the mnemonic(s). The mnemonics which can be entered as well as the type of file each represents is shown below.

Mnemonic	File Type
P	Program Load File
A	ASCII Data File
B	Binary Data File
S	System Program Load File

C=*close option*

This optional line entry item specifies how each file saved on the destination diskette is to be closed. The *close option* takes the form of a number from 0 through 5. The numbers specify the following close options.

close option	Meaning
0	Close file at EOD (end-of-data).
1	Close file at EOE (end-of-extent).
2	Close file and truncate at EOD.
4	Allocate all available diskette space to file but close file at EOD.
5	Allocate all available diskette space to file but close file at EOE.

Line Entry Item	Explanation
S= *source unit*	A decimal digit which specifies the logical unit containing the source diskette. If not specified, the default is to logical unit 0.
D= *destination unit*	A decimal digit which specifies the logical unit containing the destination diskette. If not specified default is to logical unit 1.

Default conditions exists for some line entry items omitted by the operator. The default condition for the T=*file type(s)* option is T=P,A,B,S (i.e., all types); for the C=*close option* C=0 (close file at EOD); for the S=*source unit*, unit 0; and for D=*destination unit*, unit 1. Specifying SAVE * causes SAVE to copy all files from unit 0 unit 1.

A sample of SAVE operation is shown in the following example.

```
SAVE,AFILE,BFILE,CFILE,FILE*,T=A,S=1,D=2,C=2
FILES SAVED FROM UNIT 1 TO UNIT 2 ARE:

BFILE
CFILE
FILE1
FILE2
FILE3     ALL DONE
```

Note that the operator specified a file labeled AFILE but SAVE did not list the filename among those that were saved. For purposes of this example, AFILE has been assigned a file type of P (program file). Because of its file type SAVE did not save AFILE since the operator keyboard line entry specified a T=A option, which instructed SAVE to save only files assigned an ASCII file type.

6.3.7 DISKCOPY (Copy-Diskette)

DISKCOPY copies the contents of a diskette specified as source into a diskette specified as destination. The diskette specified as destination should be an empty or scratch diskette which contains no wanted files. DISKCOPY overwrites all files, protected or not. The format of the keyboard line entry for DISKCOPY is shown below.

[DISKCOPY(*/unit no.*)][,*source unit*][,*destination unit*]↵

DISKCOPY also provides the operator a prompting message in response to the operator entering only the keyword DISKCOPY followed by a carriage return. This is shown in the following example.

```
DISKCOPY↵
ENTER SOURCE UNIT, DESTINATION UNIT
```

Line Entry Item	Explanation
DISKCOPY(*/unit no.*)	The keyword DISKCOPY initiates the program DISKCOPY.
source unit	The number of the logical unit containing the diskette to be copied.
destination unit	The number of the logical unit containing the empty or scratch diskette.

DISKCOPY indicates successful completion of its operation by printing the message: ALL DONE.

A sample of DISKCOPY operation is shown in the following example.

```
DISKCOPY,1,2↵
ALL DONE
```

6.4 THE DIAGNOSTIC UTILITIES

Four diagnostic utilities permit the operator to exhaustively test a diskette and to repair individual sectors of a diskette, to test system memory, to test diskette compatibility with other diskette drives, and to exhaustively test the ability of a diskette drive to write and read without errors: FIXD, MEMTEST, COMPAT, and EXOR.

6.4.1 FIXD (Fix-Diskette)

FIXD permits the operator to load one sector from diskette into memory; to display in hexadecimal or uncompressed ASCII one or more bytes from the sector or the entire sector on the printer or CRT; to modify one or more bytes of the sector in memory byte-by-byte or to fill a number of bytes with a hexadecimal value; and to write the modified sector in memory back to a specified location on the diskette. In addition, FIXD permits the operator to read the entire diskette once to determine if hardware read errors occur or to read from the diskette continuously to detect read errors. Each of these individual functions is selected by typing the appropriate keyword from from the following set: LOAD, GET, VERIFY, DUMP, PRINT, ASCII, MODIFY, FILL, WRITE, SCAN, EXOR, and HELP.

These keywords are specified by the operator as a line entry after FIXD has been called into execution. The format of the keyboard line entry which calls FIXD into execution is as follows: [FIXD(*/unit*)]. FIXD acknowledges a correct load by typing an asterisk (*). Thereafter, the operator can enter the command HELP, LOAD or GET. HELP prints a prompting message listing the keywords and required parameter needed to initiate each of the functions of FIXD. LOAD and GET move a sector from diskette into memory. One of these must be performed before any of the other functions can be performed.

6.4.1.1 HELP (Print-Prompting-Message)

HELP prints a prompting message to the operator which lists the keywords of the functions performed by FIXD along with the parameters required to initiate each function. To initiate this function, the operator types the keyword HELP followed by a carriage return. FIXD responds by typing the following:

```
OPTIONS ARE:
  LOAD TRACK SECTOR UNIT
  GET TRACK SECTOR UNIT
  DUMP (FROM-BYTE) (TO-BYTE)
  PRINT (FROM-BYTE) (TO-BYTE)
  ASCII FROM-BYTE ASCII-STRING
  MODIFY FROM-BYTE HEX-DATA----
  FILL (FROM-BYTE) (TO-BYTE),HEX-VALUE
  WRITE (TRACK) (SECTOR) (UNIT)
  VERIFY (TRACK) (SECTOR) (UNIT)
  SCAN UNIT
  EXOR UNIT
  HELP?
```

Each of these functions is described in the following paragraphs.

6.4.1.2 LOAD (Load-Sector)

LOAD loads into memory one sector from the track, sector and diskette logical unit specified. This function or the GET function must be performed before any of the other FIXD functions can be performed since this function moves the diskette sector contents into memory where the other FIXD functions can occur. The format of the keyboard line entry for LOAD is shown below.

[LOAD][,*track*][,*sector*][,*unit*]↵

Line Entry Item	Explanation
LOAD	The keyword LOAD informs FIXD that the load function is to be executed.
track	The number of the tract (0-76) containing the sector to be read into memory.
sector	The number of the sector (1-26) within the track to be read into memory.
unit	The logical unit number of the drive containing the diskette from which the sector is to be read.

After the keyboard line entry for the load function is made, FIXD prints an asterisk (*) to indicate that the function has been completed. An example of the keyboard line entry for LOAD is shown below.

LOAD,1,2,0↵

Sector 2 in track 1 of the diskette in logical unit 0 is loaded into memory.

6.4.1.3 GET (Get-Sector)

GET loads into memory one sector from the track, sector, and diskette logical unit specified in the same way as LOAD, however, LOAD makes one attempt to perform the load function and if an error occurs does not retry the operation. GET on the other hand will retry the operation until the load is effected or it is stopped by the operator typing any keyboard character. The format of the keyboard line entry is shown below.

[GET][,*track*][,*sector*][,*unit*]↵

Line Entry Item	Explanation
GET	The keyword GET informs FIXD that the GET function is to be executed.
track	The number of the tract (0-76) containing the sector to be read into memory.
sector	The number of the sector (1-26) within the track to be read into memory.
unit	The logical unit number of the drive containing the diskette from which the sector is to be read.

After the keyboard line entry for the get function is made, FIXD prints an asterisk (*) to indicate that the function has been completed. An example of the keyboard line entry for GET is shown below.

GET,1,25,1↵

Sector 25 in tract 1 of the diskette in logical unit 1 is loaded into memory.

6.4.1.4 DUMP (Hex-Dump)

DUMP displays all or selected bytes of the sector loaded in memory from diskette in hexadecimal on the printer or CRT. The format of the keyboard line entry for DUMP is shown below.

[DUMP](,*from-byte-no.*)(,*to-byte-no.*)↵

Line Entry Item	Explanation
DUMP	The keyword DUMP informs FIXD that the DUMP function is to be performed
from-byte-no.	The number of the byte within the sector (1-128) at which the dump is to begin.
to-byte-no.	The number of the byte within the sector (1-128) at which the dump is to conclude. This number must be the same or larger than the number specified for the start of the dump.

If the optional start- and end-byte numbers are not specified, the entire sector is displayed. An example of the display that occurs after the DUMP keyboard line entry is made is shown below. Observe that each byte in the sector is numbered from 1 through 128.

DUMP,1,128↵

BYTE	DATA															
1	CF	20	20	20	20	20	20	20	20	20	20	20	20	20	2F	20
17	20	2F	20	20	20	20	36	35	20	32	36	20	20	20	20	20
33	20	20	32	34	20	20	20	20	20	20	20	20	0D	0A	00	0A
49	11	55	4E	55	53	45	44	20	46	49	4C	45	20	4E	41	4D
65	45	53	20	20	30	37	10	0D	0A	0A	55	4E	55	53	45	44
81	20	53	45	43	54	4F	52	53	0D	0A	20	42	4F	45	20	20
97	20	4E	55	4D	42	45	52	0D	0A	00	37	30	20	30	34	20
113	20	20	20	31	37	39	0D	0A	00	55	4E	49	54	3F	20	00

6.4.1.5 PRINT (ASCII-Dump)

PRINT displays all or selected bytes of the sector loaded in memory from diskette in ASCII on the printer or CRT. The format of the keyboard line entry for PRINT is shown below.

[PRINT](,*from-byte-no.*)(,*to-byte-no.*)↵

Line Entry Item	Explanation
PRINT	The keyword PRINT informs FIXD that the PRINT function is to be performed.
from-byte-no.	The number of the byte within the sector (1-128) at which the dump is to begin.
to-byte-no.	The number of the byte within the sector (1-128) at which the dump is to conclude. This number must be larger than the number specified for the start of the dump.

If the optional start- and end-byte numbers are not specified, the entire sector is displayed. An example of the display that occurs after the PRINT keyboard line entry is made is shown below.

```
PRINT,1,80↵
        ORG    #2000

START   XRA    A
        STA    #999
        RET

        END    START
000        ORG    #2000
```

6.4.1.6 ASCII (Replace-ASCII-Data)

ASCII replaces selected consecutive bytes of the sector loaded in memory from diskette with the ASCII data specified in the ASCII keyboard line entry. The format of the keyboard line entry for ASCII is shown below.

[ASCII][,*from-byte-no*][,*ASCII string*]↵

Line Entry Item	Explanation
ASCII	The keyword ASCII informs FIXD that the ASCII function is to be performed.
from-byte-no.	The number of the byte within the sector (1-128) at which the data replacement is to begin.
ASCII string	The ASCII data to be used to replace existing data in the sector.

Once the keyboard line entry is made, FIXD prints an asterisk (*) to indicate that the ASCII replacement has been made. An example of the ASCII keyboard line entry is shown below.

ASCII,1,PROGRAM START↵

The contents of bytes 1 through 13 of the sector in memory is replaced with the ASCII string PROGRAM START. Each ASCII character occupies one byte.

6.4.1.7 MODIFY (Replace-Hex-Data)

MODIFY replaces selected consecutive bytes of the sector loaded in memory from diskette with the hexadecimal data specified in the MODIFY keyboard line entry. The format of the keyboard line entry for MODIFY is shown below.

[MODIFY][,*from-byte-no.*][,*hex data*]↵

Line Entry Item	Explanation
MODIFY	The keyword MODIFY informs FIXD that the MODIFY function is to be performed.
from-byte-no.	The number of the byte within the sector (1-128) at which the data replacement is to begin.
hex data	The hexadecimal data to be used to replace existing data in the sector.

Once the keyboard line entry is made, FIXD prints an asterisk (*) to indicate that the hexadecimal replacement has been made. An example of the MODIFY keyboard line entry is shown below.

```
MODIFY,17,AB,12,1F↵
```

The hexadecimal contents of bytes 17, 18, and 19 of the sector in memory is replaced with the hexadecimal number $AB_{(16)}$, $12_{(16)}$, and $1F_{(16)}$, respectively.

6.4.1.8 FILL (Fill-With-Hex-Value)

FILL replaces selected consecutive bytes of the sector loaded in memory from diskette with the hexadecimal value specified in the FILL keyboard line entry. The format of the keyboard line entry for FILL is shown below.

[FILL](,*from-byte-no.*)(,*to-byte-no.*)[,*hex-value*]↵

Line Entry Item	Explanation
FILL	The keyword FILL informs FIXD that the FILL function is to be performed.
from-byte-no.	The number of the byte within the sector (1-128) at which the fill replacement is to begin.
to-byte-no.	The number of the byte within the sector (1-128) at which the fill replacement is to conclude. This number must be larger than the number specified for the start of the fill.
hex-value	A one- or two-digit hexadecimal value which will be used to fill the bytes specified.

If the optional start- and end-byte numbers are not specified for the fill, the entire sector is filled with the hexadecimal value specified. Once the keyboard line entry is made, FIXD prints an asterisk (*) to indicate that the hexadecimal fill has been made. An example of the keyboard line entry for FILL is shown below.

FILL,1,16,CF↵

Bytes 1 through 16 of the sector in memory is filled with the hexadecimal value $CF_{(16)}$.

6.4.1.9 WRITE (Write-Sector-To-Diskette)

WRITE stores the sector in memory into the track and sector specified. Once the write occurs, FIXD also performs a verify to ensure that the data in memory and the data stored on the diskette is the same. The format of the keyboard line entry for WRITE is shown below.

[WRITE][,*track*][,*sector*][,*unit*]↵

Line Entry Item	Explanation
WRITE	The keyword WRITE informs FIXD that the WRITE function is to be performed.
track	The number of the track (0-76) containing the sector in which the memory sector is to be written.
sector	The number of the sector (1-26) within the track in which the memory sector is to be written.
unit	The logical unit number of the drive containing the diskette in which the memory sector is to be written.

After the keyboard line entry for the WRITE function is made, FIXD prints an asterisk (*) to indicate that the function has been completed. An example of the keyboard line entry for WRITE is shown below.

WRITE,25,1,1↵

The memory sector is written into track 25, sector 1 of the diskette in logical unit 1.

6.4.1.10 VERIFY (Verify-Memory-With-Diskette)

VERIFY compares the sector in memory with the sector on the diskette specified. If the data in memory is the same as that on diskette, FIXD prints an asterisk (*); otherwise, FIXD, prints the following message to inform the operator of the error:

BAD DATA READ

The format of the keyboard line entry for VERIFY is shown below.

[VERIFY][,*track*][,*sector*][,*unit*]↵

Line Entry Item	Explanation
VERIFY	The keyword VERIFY informs FIXD that the VERIFY function is to be performed.
track	The number of the track (0-76) containing the sector with which the memory sector is to be compared.
sector	The number of the sector (1-26) within the track with which the memory sector is to be compared.
unit	The logical unit number of the drive containing the diskette containing the diskette sector with which the memory sector is to be compared.

An example of the keyboard line entry for VERIFY is shown below.

VERIFY,25,1,1↵

The memory sector is compared with sector 1 of track 25 on the diskette in logical unit 1.

6.4.1.11 SCAN (Read-Diskette)

SCAN checks for proper operation of the diskette. It reads an entire diskette one time and concurrently monitors the diskette hardware status to determine if any read errors occur during the read operation. If no read errors occur, FIXD displays an asterisk (*); otherwise, FIXD displays one of the following messages to inform the operator of the error.

```
STATUS N2 = nn  FATAL READ ERROR        TRACK nn    SECTOR nn
STATUS N2 = nn  BAD DATA READ           TRACK nn    SECTOR nn
SOFT ERROR                              TRACK nn    SECTOR nn
```

The soft-error message states that FIXD successfully completed a read operation; however, the operation required up to five attempts before succeeding. A bad-data-read or fatal-read-error message indicates that an attempted read operation could not be successfully completed in five attempts. The significance of the 2-digit hexadecimal value after the equals (=) sign is detailed in Appendix B.

The format of the keyboard line entry for SCAN is shown below.

[SCAN][,*unit*] ↵

Line Entry Item	Explanation
SCAN	The keyword SCAN informs FIXD that the SCAN function is to be performed
unit	The logical unit number of the drive which will be checked.

An example of the diskette line entry for SCAN is shown below. FIXD reads the entire diskette in the diskette drive configured as logical unit 1.

SCAN,1 ↵

6.4.1.12 EXOR (Read-Diskette-Continuously)

EXOR checks for proper operation of the diskette. Like SCAN, it reads an entire diskette; however, unlike SCAN, EXOR continues the reading of the diskette until halted by the operator. As it reads, EXOR concurrently monitors the diskette hardware status to determine if any read errors occur during the read operation. If no read errors occur, EXOR continues reading until halted by the operator. typing any keyboard character. If an error occurs, one of the following messages is displayed to inform the operator of the error.

```
STATUS N2 = nn  FATAL READ ERROR        TRACK nn    SECTOR nn
STATUS N2 = nn  BAD DATA ERROR          TRACK nn    SECTOR nn
SOFT ERROR                              TRACK nn    SECTOR nn
```

The soft-error message states that FIXD successfully completed a read operation; however, the operation required up to five attempts before succeeding. A bad-data-read or fatal-read-error message indicates that an attempted read operation could not be successfully completed in five attempts. The significance of the 2-digit hexadecimal value after the equals (=) sign is detailed in Appendix B.

The format of the keyboard line entry for EXOR is shown below.

[EXOR][,*unit*]↵

Line Entry Item	Explanation
EXOR	The keyword EXOR informs FIXD that the EXOR function is to be performed
unit	The logical unit number of the drive which will be checked.

An example of the keyboard line entry for EXOR is shown below. FIXD reads the entire diskette in the diskette drive configured as logical unit 1 continuously.

EXOR,1↵

6.4.2 MEMTEST (Memory-Test)

MEMTEST performs two tests on the system memory. The first test checks the likelihood of the memory to drop individual bits. The second test checks the integrity of the memory refresh hardware. Each test can be inhibited by control panel switches 0 and 1: switch 0 inhibits the first test and switch 1 inhibits the second test. Once initiated, MEMTEST continues execution until halted by the operator placing the RUN/STOP switch to STOP and pressing the RESET switch.

To initiate MEMTEST, type MEMTEST followed by a carriage return. MEMTEST responds by displaying the message: MEMORY TEST. Immediately afterwards, the following sequence of numbers appears in the control panel column position indicator: 601 through 6xx (depending on memory size: a 20K system displays 601 - 606, a 64K system, 601 - 615), 621, 600, and 620 Once the sequence in the indicators completes, a short time passes and then the following message is displayed: PASS # 1. After this display occurs, MEMTEST begins its operation again by starting the sequence in the column position indicator once more.

If MEMTEST detects an error during its execution, it displays one of the following messages:

LOC $nnnn_{(16)}$ = $nn_{(16)}$ S/B $nn_{(16)}$
PARITY ERROR AT LOC $nnnn_{(16)}$ STATUS = $nn_{(16)}$

If the first message the hexadecimal number $nnnn_{(16)}$ is the location where the failure occurred. The hexadecimal number $nn_{(16)}$ following the equal sign is the data from the location specified, while the hexadecimal number $nn_{(16)}$ following S/B indicates the value which should have been at that location.

In the second message, the hexadecimal number $nnnn_{(16)}$ specifies the location in memory where the error was detected. The most significant digit of the hexadecimal number $nn_{(16)}$ can be either a zero or one. A zero in this position indicates that the parity error occurred during a processor memory fetch. A one in this position indicates the parity error occurred during a memory fetch by the direct memory access (DMA) hardware. The least significant digit of the 2-digit hexadecimal number specifies in which of 16 possible 4K clusters of memory locations the location producing the parity error is found.

6.4.3 COMPAT (Diskette-Compatibility-Test)

COMPAT tests the compatibility of diskette drives in a system with one another. The compatibility test ensures that each drive in a system writes on a diskette in such a manner that all other drives in the system can easily read the data that has been written. To test compatibility, COMPAT writes a test pattern into four tracks of a scratch (empty) diskette on the first diskette drive (logical unit) being tested, then, instructs the operator to move the scratch diskette to the next drive being tested and then writes the test pattern in four different tracks of the diskette. This sequence repeats until all drives being tested have been written into four tracks.

When COMPAT completes its writing operation, it instructs the operator to place the scratch diskette back on the first logical unit being tested. Then COMPAT reads all previously written tracks to ensure that the test pattern can be retrieved faithfully by the logical unit being tested. Once the first logical unit has read the test pattern from all previously written tracks, COMPAT directs the operator to move the scratch diskette to the next logical unit being tested and the read operation repeats. This entire sequence repeats until all logical units have read the test pattern from all previously written tracks of the scratch diskette.

The tracks in which COMPAT writes the test pattern for each of the four logical units of a system are shown below.

Logical Units	Tracks
0	0 36 45 69
1	1 37 46 70
2	2 38 47 71
3	3 39 48 72

The format of the keyboard line entry for COMPAT is shown below.

[COMPAT][,*1st unit no.*]...[,*4th unit no.*]↵

COMPAT also provides the operator a prompting message in response to the operator entering only the keyword COMPAT followed by a carriage return. This is shown in the following example.

```
COMPAT↵
ENTER UNIT NUMBERS TO BE TESTED
```

Line Entry Item	Explanation
COMPAT	The keyword COMPAT initiates the program COMPAT.
1st unit no. through *4th unit no.*	The logical unit numbers of the units to be tested. Each number specified must be separated by a comma and no number larger than 4 may be specified.

An example of the keyboard line entry for COMPAT is shown below. COMPAT is directed to test logical units 1, 2, and 3.

```
COMPAT,1,2,3↵
```

COMPAT responds to the keyboard line entry by displaying the following messages over a period of time.

```
INSTALL SCRATCH DISK IN UNIT NO. 1 AND DEPRESS SPACE BAR TO CONTINUE
INSTALL SCRATCH DISK IN UNIT NO. 2 AND DEPRESS SPACE BAR TO CONTINUE
INSTALL SCRATCH DISK IN UNIT NO. 3 AND DEPRESS SPACE BAR TO CONTINUE
```

After the first of these three messages are displayed, the operator must install a scratch diskette in logical unit 1 and press the space bar. After the second and third messages are printed, the operator must remove the scratch diskette from one logical unit and move it to the next. During this time COMPAT is writing the test pattern in different tracks of the diskette on each logical unit. Once all three messages have been displayed and the operator has responded appropriately to each, the write portion of COMPAT's operation completes.

COMPAT next prints the same three messages again in the same way it did for the write operation. This time, however, the test pattern is being read from all tracks which have been written by each logical unit tested. COMPAT does not automatically terminate its operation at conclusion of the read operation, rather it begins the read portion of its operation again. The operator must terminate

COMPAT operation by typing the ESCape key.

If an error is detected during the write portion of COMPAT's operation, the first of the two following messages is displayed. If an error is detected during the read porition of COMPAT's operation, the second of the two messages is displayed.

HEADER ERROR ON TRACK nn SECTOR nn
READ BACK ERROR ON UNIT n WRITTEN ON UNIT n TRACK nn SECTOR nn

Both messages specify the track and sector where the error occurred. In addition, the read error message also specifies the unit which failed to read and the unit which wrote the track in which the read error occurred. A header error indicates that COMPAT found no header or an error in a header of a sector in which it attempted to write the test pattern.

6.4.4 EXOR (Diskette-Exerciser)

EXOR exercises the diskette controller and the direct memory access (DMA) hardware. EXOR writes random data in a randomly selected track of a specified diskette then reads the random data from the track. This process compares the data read with that written while monitoring the diagnostic checks performed during the write and read operation to detect any error condition that might occur. EXOR executes in this manner until halted by the operator.

To initiate EXOR, type EXOR followed by a carriage return and observe that the following message is displayed. As stated in the message, load the diskette logical unit(s) to be tested with a scratch diskette and enter 0, 1, 2, and/or 3 followed by a carriage return to select logical units 0, 1, 2, and/or 3 respectively.

```
:EXOR↵
INSTALL SCRATCH DISK(S) THEN ENTER UNIT NUMBER(S)
```

During execution, EXOR permits the operator to select one of two types of reports on a diskette drive being tested. The first is a running total of soft, data, and header errors. This report is displayed when requested by the operator. On systems equipped with a CRT, the running total is displayed continuously on the CRT. The second report is a log report displayed each time an error is detected.

To display the running total report on a system equipped with a printer, type R and observe that the following report is printed. The report lists the current total of write and verify operations performed along with a total of software (soft) and hardware (data and header) errors that have occurred for each logical unit being tested.

```
00/00/00  01:04:54
          UNIT 0   UNIT 1   UNIT 2   UNIT 3   UNIT 4   UNIT 5   UNIT 6   UNIT 7
WRITES                       261
VERIFYS                      260
ERRORS
SOFT
DATA
HEADER
```

A soft error indicates that EXOR successfully complete a write or read operation; however, the operation required up to five attempts before succeeding. A data error indicates that an attempted write or read operation could not be successfully completed in five attempts. A header error indicates that EXOR found no header or an error in a header of a sector in which it attempted to write.

To display the log report, type an L and observe that the following report is displayed. To turn the report off, type Q. EXOR responds to the operator input with the first message which follows; thereafter, it produces one of the next three messages listed anytime an error is encountered. The last message is printed after the operator turns off the log report.

```
LOG ON

UNIT n STATUS N2 = xx  BAD DATA READ          TRACK xx   SECTOR xx  date time
UNIT n STATUS N2 = xx  FATAL READ ERROR       TRACK xx   SECTOR xx  date time
UNIT n SOFT ERROR                             TRACK xx   SECTOR xx  date time

LOG OFF
```

The soft-error message indicates the same error as that described for the running total report. The bad-data-read and fatal-read-error message specify the same type of error as the data error described in the running total report. The significance of the 2-digit hexadecimal value after the equals (=) sign is detailed in Appendix B.

To cause EXOR to test another diskette logical unit, type a U and observe that EXOR displays the following message. Load the logical unit(s) to be tested with a scratch diskette and enter the appropriate logical unit numbers followed by a carriage return on the keyboard to continue.

```
INSTALL SCRATCH DISK(S) THEN ENTER UNIT NUMBER(S)
```

To halt EXOR operation, type any keyboard character except Q, L, U, or R and observe that EXOR prints a log report on a system equipped with a printer followed by a colon to indicate a return to Command Processor. On CRT systems, the log report is updated and a colon is displayed.

6.4.5 TIME and TIME SET (The Clock Utilities)

The clock utilities permits the operator to display the time of day and to set the date (month, day, and year) and time-of-day (hour, minute, and second).

To display day and date, type the keyword TIME followed by a carriage return and observe that the time and date are displayed.

To set date and time enter the keywords TIME SET followed by a carriage return and observe that the following message is printed.

TIME? (HH,MM,SS)

Insert the hour, minute, and second of time each separated by a comma and then enter a carriage return. Observe that the following message is printed next.

DAY? (MM,DD,YY)

Insert the month, day, and year each separated by a comma and then enter a carriage return. The time and day are now in the system.

SECTION 7 - PROGRAM DEBUGGING

7.1 INTRODUCTION

The System 3200 Command Processor permits the operator to load and debug programs written in System 3200 Assembly Language or DACL. However, the methodology used to debug assembly language programs differs from that used to debug DACL programs. This difference exists because assembly language source code produces one machine instruction for each line of source code (comment lines excluded). DACL source code produces an object code which, in turn, must be interpreted by the system program INT(erpreter). Thus a DACL program must be debugged as it is interpreted (executed) by the interpreter (INT). This discussion is divided into assembly language and DACL debugging.

7.2 ASSEMBLY LANGUAGE PROGRAM DEBUGGING

Command Processor enables an operator to load a program to be debugged into microcomputer memory without causing the program to execute. Command Processor enables the operator to dump the contents of one or more memory locations, to store data in any memory location, to enter data in any of the microcomputer registers, and to set a breakpoint in the program being debugged. One other program debugging service provided by the command processor is that it allows the operator to call into execution or to continue executing the program being debugged. The keyboard line entry format which commands each function is contained in the following paragraphs.

7.2.1 Load Program for Debugging (HEX)

The LOAD command calls a program to be debugged into microcomputer memory without causing the program to execute. The operator enters the keyword LOAD followed by the name of the program to be debugged. The keyboard line entry format is shown below.

[LOAD][,*program name*]↵

Line Entry Item	Explanation
LOAD	Keyword that specifies the load-program function.
program name	The name of the program to be debugged.

7.2.2 Dump Memory (DMP)

The DMP command displays the contents of one or more memory locations on the system printer. The operator enters the keyword DMP followed by a single address or an initial address (where the dump is to begin) and a terminal address where the dump is to conclude). The keyboard line entry format is shown below. To terminate a dump operation in progress, type any key on the keyboard.

[DMP][,*address 1*](,*address 2*)↵

Line Entry Item	Explanation
DMP	Keyword that specifies the dump-memory function.
address 1	Any single memory address from $0000_{(16)}$ to $FFFF_{(16)}$. This entry is mandatory. If this is the only address specified, a dump of two memory location occurs.
address 2	Any memory address from $0000_{(16)}$ to $FFFF_{(16)}$. This address represents the last address in the series of addresses from address 1 to address 2 that are being displayed on the printer or CRT.

7.2.3 Enter Data in Microcomputer Memory (HEX)

The HEX command inserts data into any microcomputer memory location. The operator enters the keyword HEX followed by the hexadecimal representation of the address in which data is to be stored followed by the hexadecimal representation of the actual data to be stored. The keyword line entry format is shown below.

[HEX][,*address*][,*data 1, data 2,... data n*]↵

Line Entry Item	Explanation
HEX	Keyword that specifies the enter-memory function.
address	Any memory location from $0000_{(16)}$ to $FFFF_{(16)}$. Observe that memory location $0000_{(16)}$ to $1000_{(16)}$ can be specified. These locations contain the object code for the system Control Program, CP. Thus ensure that locations in this area of memory are not inadvertently destroyed.
data 1, data 2,... data n	A maximum of two contiguous hexadecimal digits may be specified for each individual data entry. The two hexadecimal digits produce eight bits of data which are stored in the address specified in the line entry. If more than one data entry is specified, each additional data entry is stored in the next higher memory location.

7.2.4 Enter Data in Microcomputer Registers (SET)

The SET command inserts data into any one of the microcomputer registers. The operator enters the keyword SET followed by the mnemonic of the register to receive the data followed by the data to be entered in the register. The keyboard line entry format is shown below.

[SET][,*register*][,*data*]↵

Line Entry Item	Explanation
SET	Keyword that specifies the enter-data function.
register	Any of the following microcomputer registers can be specified: A, B, C, D, E, F, H, L, and P. The mnemonic F specifies the 5-bit status register while P refers to the program counter. For an explanation of the microcomputer registers refer to the Ranger System Assembly Language Programming Manual, SYSPG02P.
data	A maximum of two contiguous hexadecimal digits may be specified. The two hexadecimal digits produce eight bits of data which are stored in the specified register.

7.2.5 Set Breakpoint (BKP)

The BKP command sets up to five breakpoints at specific program addresses in the program being debugged, i.e., the operator can enter five BKP commands each specifying a different program address. Each BKP command saves the breakpoint address and the operation code contained in the address in a table. Next the BKP command places an assembly language restart 1 instruction in the breakpoint address. When the breakpoint program address is reached during execution of the debugged program, the restart 1 instruction causes a branch to location $8_{(16)}$. Note that a breakpoint must be set into a program address containing an instruction operation code, otherwise, the breakpoint will never be executed. A restart routine located at location $8_{(16)}$ removes the restart 1 instruction from the breakpoint address and transfers from the table to the address the original operation code of the location. Thus a breakpoint once executed is removed from the program being debugged.

The restart routine also produces a printout of the state of the microcomputer registers at the time of breakpoint and causes the Command Processor to begin executing in the microcomputer. A sample printout/display of processor registers is shown below.

```
P=2A3A    A=0A  F=56  B=00  C=04  D=34  E=44  H=2A  L=51    SP=7FFA
:
```

NOTE

If it becomes necessary to reset (RESET button pressed) the System 3200 during a debug operation, the breakpoint addresses and associated operation code are cleared from the restart 1 table; however, the restart 1 code still exists in the program being debugged. To recover, the operator can reload the program and breakpoints or reload the breakpoints and resume operation with the program which aborted. The second action assumes that the system Control Program is intact.

The keyboard line entry format to set a breakpoint is shown below.

[BKP](,*address*)↵

Line Entry Item	Explanation
BKP	Keyword that specifies the set-breakpoint function.
address	The address in hexadecimal of the location at which the breakpoint is to be set. If no address is specified, all breakpoints are cleared.

7.2.6 Execute Debugged Program (G)

The G command causes a program being debugged to resume operation after a breakpoint has been encountered or after the operator has set the P register using the SET command. In addition, the G command calls into execution the program loaded into the microcomputer by the LOAD keyword. The keyboard line entry format required to call a debugged program into execution is shown below.

[G](,*parameter 1*,... *parameter n*)↵

Line Entry Item	Explanation
G	Keyword that specifies the program being debugged is to be executed.
parameter 1,... *parameter n*	Any parameter required by the program to execute.

7.3 DACL PROGRAM DEBUGGING

Because DACL source statements are interpreted, the command processor commands to debug a DACL program are limited to the HEX, DMP, and G commands. Each of the functions are discussed and an example is provided to illustrate the DACL debugging operation.

7.3.1 Debugging Function

The HEX command sets or clears a breakpoint in a DACL program, starts a trace, and inserts data into any microcomputer memory location. A DACL breakpoint is set by loading memory location $0004_{(16)}$ with an address corresponding to an individual statement of a DACL program. Loading location $0004_{(16)}$ with zero clears the breakpoint. A DACL trace may be started by loading address $0006_{(16)}$ with a zero, placing any value other than zero in the location halts the trace. The HEX command can also be used to perform the same functions as described in paragraph 7.2.3. The DMP and G commands are the same for both DACL and assembly language debugging. See paragraphs 7.2.2 and 7.2.6, respectively.

7.3.2 Example of DACL Program Debugging

Figure 7-1 shows the compilation of sample program DEBGFIL1 and subsequent creation of program file DEBGFIL4. The DACL compilation call is made by responding to prompting statements printed by DACL. This permits the operator to request DACL to print/display the DACL code, a helpful printout/display when debugging a DACL program.

Figure 7-2 shows the printout resulting from the DACL compilation. The leftmost column of numbers are the program statement numbers. The next column contains memory locations containing the DACL code, the third column from the left is the DACL code represented in hexadecimal. The two remaining columns contain the DACL program statements. The internal representation of a DACL program is described in the DACL Programming Reference Manual.

```
:DACL
SOURCE FILE NAME: DEBGFIL1
UNIT NUMBER: 1
OBJECT FILE NAME: DEBGFIL4
UNIT NUMBER: 1
PRINT THE OUTPUT? YES
DISPLAY THE OUTPUT? NO
PRINT THE CODE? YES
PRINT THE HEADING:
DACL DEBUG DEMONSTRATION
```

Figure 7-1. DACL Compilation Call

```
PAGE 01  DACL DEBUG DEMONSTRATION

  1.  3476   80 30 83          A        FORM "0"
  2.  3479   80 31 83          ONE      FORM "1"
  3.  347C   80 33 83          THREE    FORM "3"
  4.  347F   D2 81 80          LOOP     ADD ONE TO A
  5.  3482   D1 80 82                   COMPARE A TO THREE
  6.  3485   D9 00 81                   GOTO LOOP IF NOT EQUAL
  7.  3488   64 41 20 3D               PRINT "A = ",A,*10,"ONE = ",ONE,*20,"THREE = ",THREE
  7.  348C   20 80 12 09
  7.  3490   4F 4E 45 20
  7.  3494   3D 20 81 12
  7.  3498   13 54 48 52
  7.  349C   45 45 20 3D
  7.  34A0   20 82 FF
  8.  34A3   60 41 20 3D               KEYIN "A = THREE"
  8.  34A7   20 54 48 52
  8.  34AB   45 45 FF
  9.  34AE   60 50 52 4F               KEYIN "PROGRAM COMPLETE"
  9.  34B2   47 52 41 4D
  9.  34B6   20 43 4F 4D
  9.  34BA   50 4C 45 54
  9.  34BE   45 FF
 10.  34C0   5E                       STOP
      34C1   5E                       STOP

      34C2   7F 34           LOOP

      34C4   76 34           A
      34C6   79 34           ONE
      34C8   7C 34           THREE

      3402   34 C2 34 C4
      3406   34 7F 34 00
      340A   00 00 00 00
      340E   00 00 00 00
      3412   00 00 00 00
      3416   00 00 00 00
      341C   00 00 00 00
      3420   00 00 00 00
      3424   00 00 00 00
      342A   00 00 00 00
      342E   00 00 00 00
      3432   00 00 00 00
      3438   00 00
      343C   00 00
      3440   00 00 00 00
 0 ERRORS...DONE

:
```

Figure 7-2. DACL Listing with Object Code

Figure 7-3 shows a sample DACL debug operation. Observe that before the INT is called, a DACL breakpoint is set at location $3485_{(16)}$. The breakpoint is set by loading address $4_{(16)}$ with address $3485_{(16)}$ using the HEX command. It is essential to set at least the first DACL breakpoint prior to calling INT. Next INT is called and the name of the program being debugged DEBGFIL4 is provided. Thereafter INT executes program DEBGFIL4 until the breakpoint at $3485_{(16)}$ is reached. There, INT halts execution and outputs the contents of the P register for the DACL program along with the state of condition flags. An explanation of the four condition flags is contained in the DACL Programming Reference Manual. INT also outputs the statement "RST 1 EXECUTED" followed by the state of all microcomputer registers at the time the breakpoint occurred. These describe the state of INT and are of no concern to the DACL programmer.

At the breakpoint the microcomputer returns to the system Command Processor. The operator enters a zero at location $6_{(16)}$ to turn on a trace, changes the constant located at address $347A_{(16)}$ to a 32 (an ASCII 2), and dumps the contents of DACL location $3476_{(16)}$. (Refer to figure 7-2.) Loading the 32 (an ASCII 2) in DACL location $347A_{(16)}$ replaces the original 31, an ASCII 1, with an ASCII 2, thus changing the value of "one" to 2. This illustrates the type of change to memory that can be carried out using the HEX debugging command. Dumping (DMP) location $3476_{(16)}$ causes a display of both location $3476_{(16)}$ and $3477_{(16)}$.

Typing G causes the program being debugged to continue execution. However, since, a trace was started, each DACL location is output along with a display of the four INT condition flags. This list of DACL program addresses continues until the next breakpoint is reached or the DACL program completes. In the example, the program contains a loop and the original DACL breakpoint is once again encountered. This causes INT to halt program execution and to display the three lines of data previously described. Observe that one of the four flags, EQ, now contains a 1 in place of the 0 it previously held. This indicates that the condition in statement 6 (Figure 7-2) will not be satisfied and that an exit will be made from the loop. A dump of location $3476_{(16)}$ shows that the value of "A" does equal three. The DACL breakpoint is, next, removed by loading (HEX) address $4_{(16)}$ with 00,00, and the trace is turned off by placing (HEX) a 1 in address 6(16). Typing G, allows the program to complete.

```
:HEX 4,34,85

:INT
/D A C L  1.1
PROGRAM NAME: DEBGFIL4

P=3485  ES0 EQ0 LS0 OV0
RST 1 EXECUTED

P=2A3A    A=0A  F=56  B=00  C=04  D=34  E=44  H=2A  L=51    SP=3FFA

:HEX 6,0

:HEX 347A 32

:DMP 3476
3476    80  31

:G

P=347F  ES0 EQ0 LS0 OV0

P=3482  ES0 EQ0 LS0 OV0

P=3485  ES0 EQ1 LS0 OV0
RST 1 EXECUTED

P=2A3A    A=0A  F=56  B=00  C=04  D=34  E=44  H=2A  L=51    SP=3FFA

:DMP 3476
3476    80  33

:HEX 6,1

:HEX 4,00,00

:G
A = 3    ONE = 2    THREE = 3
A = THREE
PROGRAM COMPLETE
/D A C L  1.1
PROGRAM NAME:
```

Figure 7-3. An Example of DACL Debugging

APPENDIX A - PRINTWHEEL AND RIBBON REPLACEMENT

TO BE SUPPLIED

APPENDIX B - STATUS MESSAGES AND INDICATIONS

B.1 GENERAL

The system 3200 provides the operator two groups of status messages and one set of status indications. One group of status message, generated by the Command Processor, describe the condition of the system during execution of a program call. A second group of messages, generated by the disk input/output device driver, I/O:FDK, a system software module describes the status of the disk hardware and software. The status indications appear in the line position indicator lights on the control panel and are generated by the resident Control Program (CP).

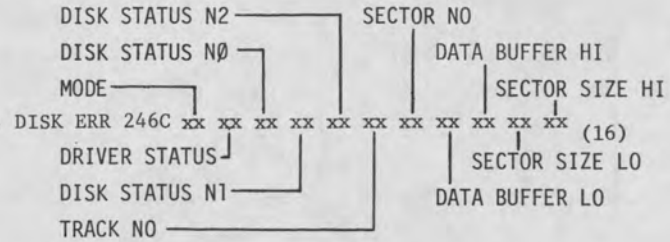
B.2 PROGRAM LOAD STATUS MESSAGES

The program load status message describes the condition of the system during execution of a program call by Command Processor. These messages are listed in Table B-1.

B.3 DISK STATUS MESSAGES

The disk status messages describe the condition of the disk hardware and software. The message is printed when a disk operation has not completed successfully. The format of the printed message is shown below along with a sample printout. The variables contained in the message are described in Figure B-1.

MESSAGE FORMAT



SAMPLE MESSAGE

DISK ERR 246C 21 A0 21 4C 20 00 00 21 AA 24 0E

Table B-1. Program Load Status Message

Message	Probable Cause	Corrective Measure
RST EXECUTED	A restart 1 instruction has been executed. This implies that a program has branched out of bounds, i.e., to a nonexistent memory address. This message also indicates that a breakpoint address has been reached in debug operation. The machine state is printed along with the message.	Examine the contents of the stack using the DMP command of command processor debug operation to determine previous program flow. Reinitialize the system if not performing a debug operation.
LOAD ERROR xx	Error occured while attempting to load a program.	
xx = $02_{(16)}$	Logical unit unavailable.	Activate logical unit using command processor UNIT command.
$03_{(16)}$, $15_{(16)}$	Disk input/output error occurred during load.	Retry the program call.
$0A_{(16)}$	Program does not exist on diskette.	Ensure program exists on diskette.
$20_{(16)}$	Premature end of file.	Reassemble program.
$22_{(16)}$	Block count error.	Reassemble program.
$23_{(16)}$	Block type error.	Reassemble program.
$24_{(16)}$	Checksum error.	Retry the program call.
LOGICAL UNIT x IS NOT AVAILABLE	A diskette operation was attempted on an undefined logical unit.	Define logical unit using command processor UNIT command.
DISKETTE IN UNIT x IS WRITE PROTECTED	A write operation was attempted on a diskette with a write protect hole.	Impossible to write on this diskette unless write protect hole is covered.
READY UNIT x	Logical unit x is not ready.	Ensure diskette is inserted properly, wait 2 seconds, and depress space bar to abort, then depress ESC key.

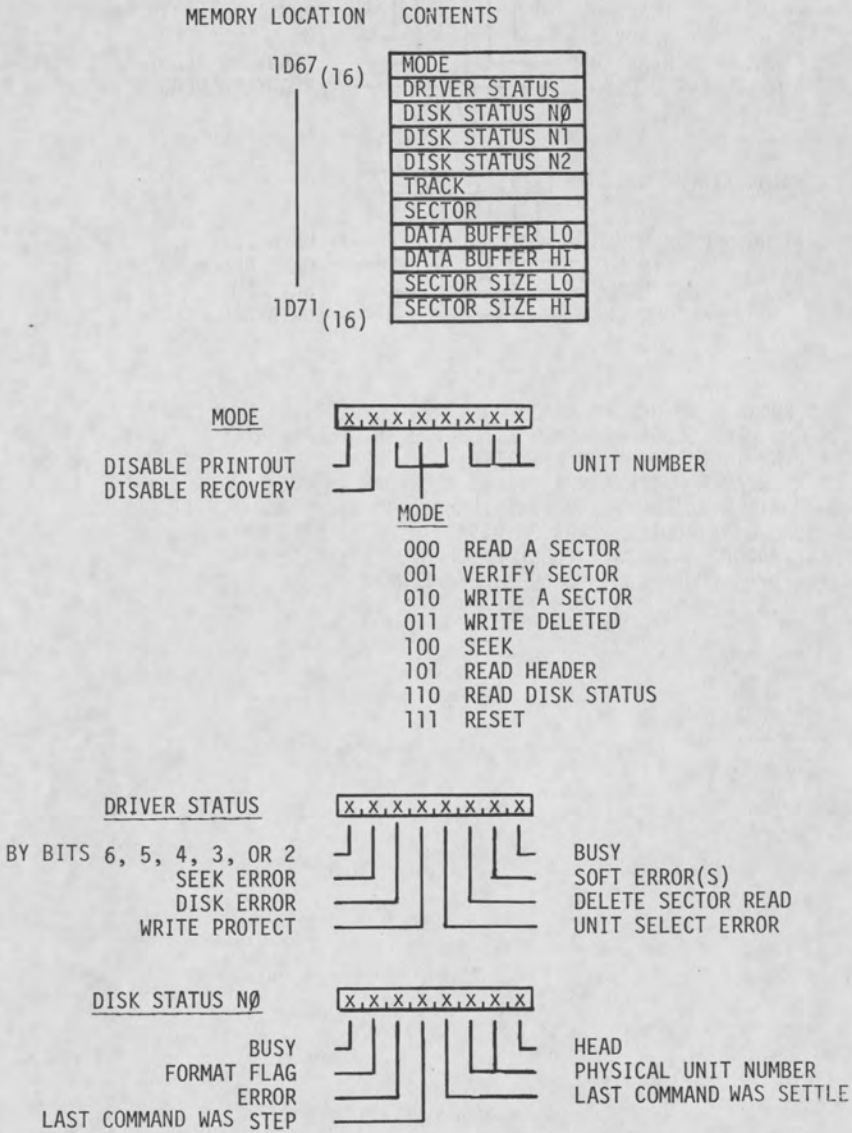
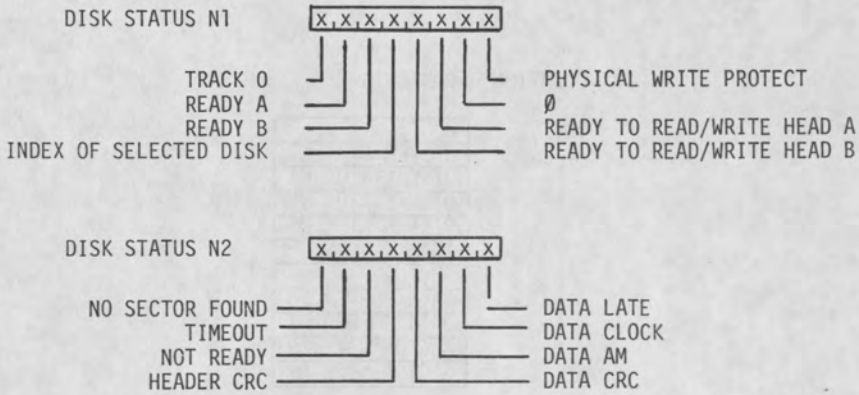


Figure B-1. Disk I/O Errors (1 of 2)



NO SECTORS FOUND - NO HEADER MATCH FOR FOUR REVOLUTIONS OF DISKETTE.
TIMEOUT - ONE-HALF SECOND ELASPED WITHOUT DISK OPERATION COMPLETING.
NOT READY - NO INDEX FOUND ON DISKETTE.
HEADER CRC - HEADER MATCHED BUT CYCLIC REDUNDANCY CHECK ERROR EXISTS IN HEADER.
DATA CRC - DATA READ FROM DISK CONTAINS CYCLIC REDUNDANCY CHECK ERROR.
DATA AM - BAD DATA ADDRESS MARK ON DISKETTE.OR DELETED DATA AM.
DATA CLOCK - ADDRESS OR DATA CLOCK PATTERN ON DISKETTE BAD.
DATA LATE - DMA REQUEST NOT HONORED BEFORE NEXT TRANSFER TIME.

Figure B-1. Disk I/O Errors (2 of 2)

B.4 SYSTEM STATUS INDICATIONS

The system status indications are displayed in the line position indicator lights on the system control panel. There are four groups of indications that appear in the indicator lights: IPL errors, parity errors, unknown interrupt errors, and printer errors. IPL errors produce one of the following numbers in the line position indicator lights: 900 - 915, 921, 970 - 974, and 999. Parity error produce a number between 700 to 715 and 780 to 795. Unknown interrupt errors produce a number between 800 and 877. Finally, printer error produce a number between 600 and 607. The significance of each error indication is detailed in Table B-2.

Table B-2. System Status Indications

Number	Explanation
0	Processor not running.
999 with display in lights 0 - 15.	The bootstrap routine in read-only memory (ROM) which performs the IPL is moved from ROM into random-access memory (RAM) while the number 999 is displayed. The system halting with 999 displayed and an unknown display in light indicators 0 - 15 indicates that a failure occurred during this movement. If the system halts with lights 0 - 15 empty, the routine moved from ROM to RAM successfully; however, failed to initiate properly.
900 - 915 with display in lights 0 - 15.	Indicates an error during execution of the memory test. Light indicators 0 - 7 contain the bit pattern that failed in the memory test while lights 8 - 15 contains the high order eight bits of the memory address where the failure occurred.
921	Indicates an error during execution of the direct memory access memory refresh phase of the memory test. Light indicators 0-7 contain the bit pattern that failed while lights 8-15 contain the high order eight bits of the address where the pattern failed.
970	Indicates that the diskette drive containing the diskette to be loaded is not ready: access door not completely closed, diskette not in receptacle properly, or a hardware failure. Lights 8 - 15 contain the diskette N1 status word. See Appendix B for a description of this word.
971	Indicates that the bootstrap routine is unable to locate track 0 on the diskette. If light indicators 0 - 7 are empty, lights 8 - 15 contain the diskette N1 status word. See Appendix B for a description of the word. If lights 0 - 7 contain data, the data contained is the N0 status word while lights 8 - 15 contain the N2 status word. See Appendix B for a description of these two words.

Table B-2. System Status Indications (cont)

Number	Explanation
972	The bootstrap routine has issued a step command to move the diskette read/write head off of track 0 and onto another track; however, the command is not being carried out by the hardware. Light indicators 0 - 7 contain the N0 status word while lights 8 - 15 contain the N1 status word. See Appendix B for a description of these two words.
973	The bootstrap routine attempted to read a sector from the diskette in which it expected to find binary data but did not. This error condition could be caused by attempting to read a diskette which does not contain FDOS.
974	The bootstrap routine attempted to read a sector from the diskette and found the checksum it calculated was different than that stored on the diskette for the sector. This could indicate that the diskette itself is bad or that the drive is malfunctioning.
84*x*	Any error indication in the 840 through 847 range indicates an unknown interrupt error. The error can occur if the system CRT has failed. The variable x represents one of eight different hardware status conditions at the time of the interrupt.
7*xxx*	Any 700 error indicates a parity error. The *xx* can assume a value from 0 through 15 which corresponds to the 4K section of memory in which the parity error occurred.
601	The printer is out of ribbon or its cover is open
602	The printer is out of paper.
603	Both the 601 and 602 conditions exist in the printer.
604	The printer is in a printer-check condition, i.e, it is up against a stop or cannot execute the operation it has been commanded to perform.

Table B-2. System Status Indications (cont)

Number	Explanation
605	Both the 601 and 604 conditions exists in the printer.
606	Both the 602 and 604 conditions exists in the printer.
607	The 601, 602, and 604 conditions exists in the printer.

APPENDIX C - HOW TO PREPARE A DISK FOR USE IN SYSTEM 3200

To prepare a diskette for operation in the System 3200 requires two operation: execution of the FORMAT utility followed by execution of the INIT utility. FORMAT configures the tracks and sectors of a diskette for operation in the System 3200 while INIT writes a null file indication in each sector of the diskette file directory (track 0, sectors 8-26 and all sectors of track 1).

FORMAT contains a default condition which will write a standard format on a diskette. To format a diskette in this standard format type the keyword FORMAT followed by a comma delimiter followed by the logical unit number of the diskette drive containing the diskette being formatted. See the example below:

FORMAT,1

This keyboard line entry causes the utility FORMAT to format the entire diskette in logical unit 1 with a standard format. Once the keyboard line entry is made, FORMAT responds with the following message.

DISK FORMATTER, ENTER Y TO CONTINUE

To this the operator must respond with an upper case Y. Thereafter FORMAT complete the format process and indicates its completion by producing the following message

ALL DONE

Once FORMAT completes, the operator can then initialize the formatted diskette. This is accomplished by entering the keyword INIT followed by a comma followed by the number of the logical unit containing the diskette to be initialized.

INIT,1

This keyboard line entry causes the utility INIT to initialize the diskette file directory of the diskette in the logical unit specified. Once the keyboard line entry is made, INIT responds with the following message.

DISK INITIALIZER, ENTER Y TO CONTINUE

To this the operator must respond with an upper case Y. Thereafter INIT completes the initialization process and indicates its completion by producing the following message:

ALL DONE

The diskette is now ready for use in the System 3200.