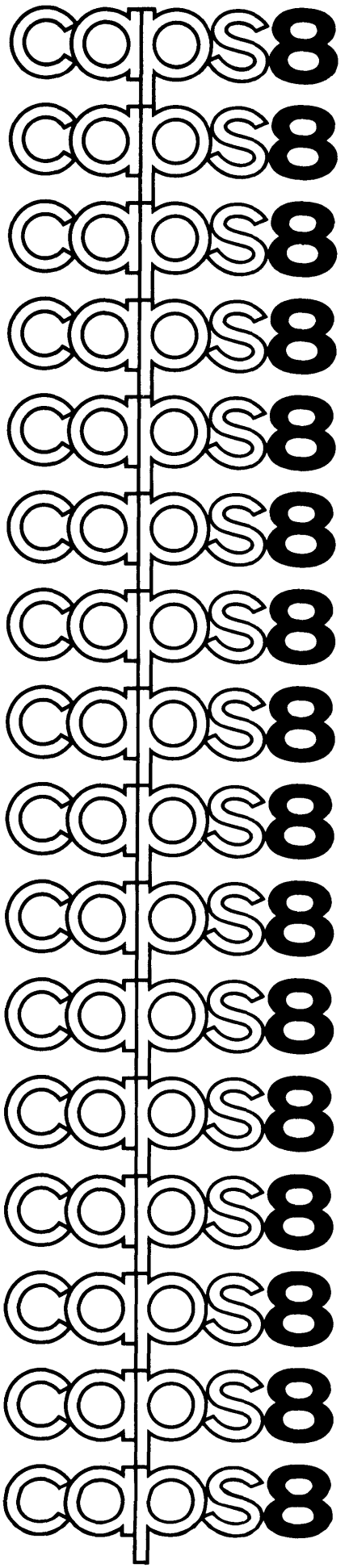
digital



cassette programming system

users manual

digital equipment corporation

DEC-8E-OCASA-B-D

CASSETTE PROGRAMMING SYSTEM
USER'S MANUAL

For additional copies, order No. DEC-8E-OCASA-B-D from Software Distribution Center, Digital Equipment Corporation, Maynard, Massachusetts 01754

First Printing, March, 1973
Revised September, 1973
Printed July, 1974

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this manual.

The software described in this document is furnished to the purchaser under a license for use on a single computer system and can be copied (with inclusion of DIGITAL's copyright notice) only for use in such system, except as may otherwise be provided in writing by DIGITAL.

Digital Equipment Corporation assumes no responsibility for the use or reliability of its software on equipment that is not supplied by DIGITAL.

The HOW TO OBTAIN SOFTWARE INFORMATION page, located at the back of this document, explains the various services available to DIGITAL software users.

The postage prepaid READER'S COMMENTS form on the last page of this document requests the user's critical evaluation to assist us in preparing future documentation.

The following are trademarks of Digital Equipment Corporation:

CDP	DIGITAL	INDAC	PS/8
COMPUTER LAB	DNC	KA10	QUICKPOINT
COMSYST	EDGRIN	LAB-8	RAD-8
COMTEX	EDUSYSTEM	LAB-8/e	RSTS
DDT	FLIP CHIP	LAB-K	RSX
DEC	FOCAL	OMNIBUS	RTM
DECCOMM	GLC-8	OS/8	RT-11
DECTAPE	IDAC	PDP	SABR
DIBOL	IDACS	PHA	TYPESET 8
			UNIBUS

CONTENTS

APPENDICES

TABLES

ILLUSTRATIONS

CHAPTER 1
THE CASSETTE PROGRAMMING SYSTEM

1.1 INTRODUCTION TO A CASSETTE STORAGE SYSTEM

The PDP-8 Cassette Programming System (CAPS-8) is a small programming system for the PDP-8/E (8/M or 8/F) computer and is designed around the use of cassettes for program storage, rather than DECtape, paper tape or disk storage. CAPS-8 replaces paper tape procedures completely. The MI8-E Hardware Bootstrap initially loads the Cassette Keyboard Monitor into memory; with the use of the Monitor all file transfers and program loading and storage is done via cassette. Cassettes are more convenient and reliable and much easier to use than paper tape, and in addition, cut the time involved in loading and storing programs using paper tape by almost one half.

CAPS-8 provides the user with a Keyboard Monitor, I/O facilities at the Monitor level, and a library of System Programs, including a machine language assembler, an editor, and a higher-level programming language.

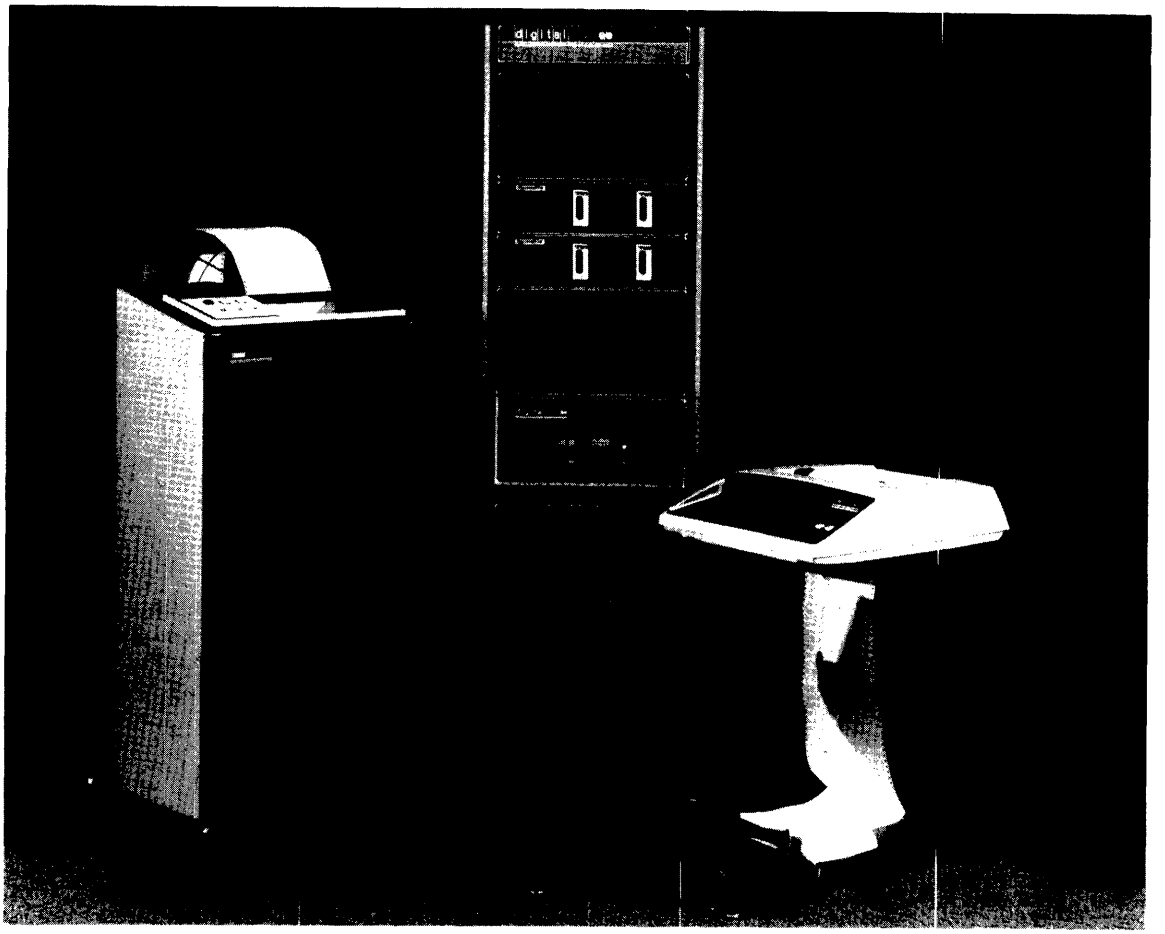


Figure 1-1 Cassette Programming System

1.1.1 Hardware Components

The Cassette Programming System is built around a PDP-8/E, 8/M, or 8/F computer with a minimum of one TU60 dual cassette unit, a console terminal (LA30 DECwriter, LT33 or LT35 Teletype, or VT05 DECterminal), and 8K of memory. A line printer is optional.

1.1.2 Software Components

A brief description of the software package available with the Cassette Programming System follows. Each program is discussed in greater detail later in the manual.

1. MONITOR - The Keyboard Monitor provides communication between the user and the Cassette System Executive Routines by accepting commands from the console terminal keyboard. The commands allow the user to run system and user programs, save programs on cassette, and obtain directories of cassettes.

2. Symbolic EDITOR - The EDITOR allows the user to modify or create source files for use as input to language processing programs such as BASIC and PALC. The EDITOR contains powerful text manipulation commands for quick and easy editing.

3. PALC Assembler (Program Assembly Language--Cassette) - PALC accepts source files in the PAL machine language and generates absolute binary files as output. These files can then be loaded and executed using Monitor commands.

4. BASIC - BASIC provides a higher-level programming language which is easy to learn and use. It includes such language features as user-coded functions, data files on cassette, and program chaining.

5. System Copy (SYSCOP) - SYSCOP allows the user to transfer files from one cassette to another, giving him the ability to make multiple copies of a cassette and "clean up" full cassettes so that they may become available for future use.

1.2 WHAT IS A CAPS-8 CASSETTE?

A CAPS-8 cassette is a magnetic tape device much like that used in a cassette tape recorder. The tape itself and the reels it is wound on are enclosed inside a rectangular plastic case (see Figure 1-2), making handling, storage, and care of the cassette convenient for the user.

On either end of one side of the cassette are two flexible plastic tabs called write protect tabs (see A in Figure 1-2). There is one tab for each end of the tape; since data should only be written in one direction on the tape, the user will need to be concerned with only the tab which is specifically marked on the cassette label. Depending upon the position of this tab, the user is able to protect his tape against accidental writing and destruction of data. When the tab is pulled in toward the middle of the cassette so that the hole is uncovered, the tape is write-locked; data cannot be written on it and any attempt to do so will result in an error message. When the tab is pushed toward the outside of the cassette so that the hole is covered, the tape is write-enabled and data can be written onto it. Data can be read from the cassette with the tab in either position.

The bottom of the cassette (B in Figure 1-2) provides an opening where the magnetic tape is exposed. The cassette is locked into position on a TU60 cassette unit drive so that the tape comes in contact with the read/write head through this opening.

Both ends of the magnetic tape in a cassette consist of clear plastic leader/trailer tape; this section of the tape is not used for information storage purposes, but as a safeguard in handling and storing the cassette itself. Since cassette tape is susceptible to dust and fingerprints, the leader/trailer tape should be the only part of the tape exposed whenever the cassette is not on a drive.

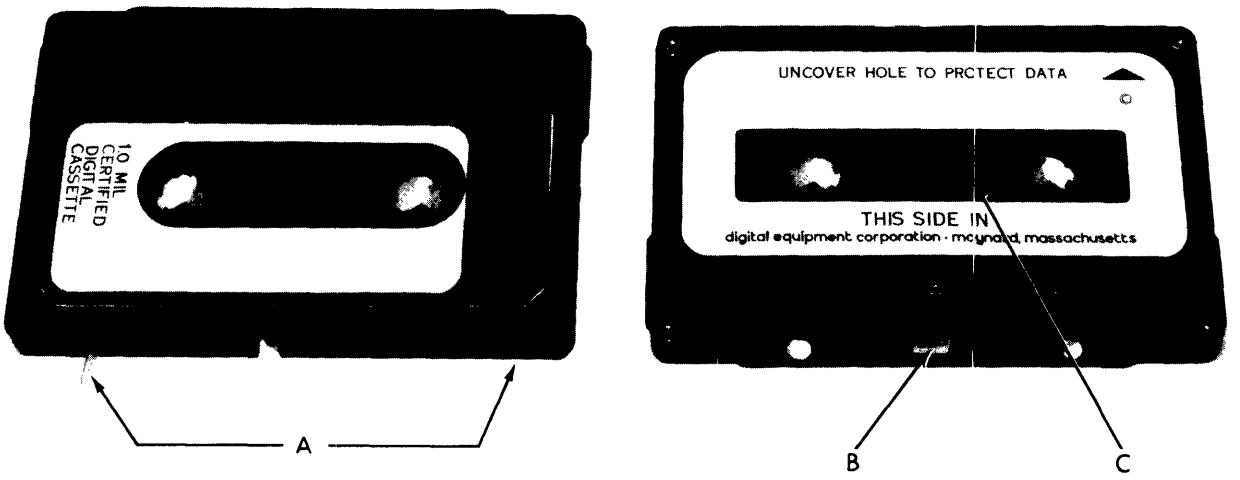


Figure 1-2 CAPS-8 Cassette

1.2.1 The Format of a Cassette

A cassette is formatted so that it consists of a sequence of one or more files. Each file is preceded and followed by a file gap. (A gap in this sense is a set length of specially coded tape.) All cassettes must start with a file gap; any information preceding the initial file gap is unreliable. A file consists of a sequence of one or more records separated from each other by a record gap. The first record of a file is called the file header record and contains information concerning the name of the file, its type, length, and so on. (Chapter 2 provides more information concerning header records.) A record generally contains 128 (decimal) characters of information; there are approximately 600 records per cassette tape.

Records consist of a sequence of one or more cassette bytes; a byte in turn consists of eight bits each representing a binary zero or one. Characters and numbers are stored in bytes using the standard ASCII character codes (see Appendix A) and binary notation.

The number of records of information on a cassette tape may be estimated by the user. On the outside of the cassette case is a clear plastic window (C in Figure 1-2). Along the bottom of this window is a series of marks; each mark represents about 50 inches of magnetic tape. Knowing that approximately 2 records fit on an inch of tape, the user is able to make a reasonable guess as to the length of tape and number of records available for use. By simply glancing at the width of the tape reel showing in the window, the user can tell quickly if he is very close to the end. Since he is given no advance warning of a full tape condition, the user must visually keep track of the length of tape he has available. Should the tape become full before his file transfer has completed, another cassette must be substituted, and the transfer or output operation must be restarted.

1.2.2 The Sentinel File

The last file on a cassette tape is called the sentinel file. This file consists of only a file header record and represents the logical end-of-tape. A zeroed or blank cassette tape is one consisting of only the sentinel file.

1.3 THE SYSTEM CASSETTE

The software discussed in Section 1.1 is provided to the user on a single cassette called the System Cassette. This is the cassette on which the entire CAPS-8 System resides, and it is utilized for all system functions. The System Cassette must always be mounted on drive 0; drive 0 serves as the default device when the user fails to specify another.

NOTE

Each TU60 dual cassette unit has two drives. The drive on the left is always odd-numbered and the drive on the right even-numbered; thus, drive 0 will be the left drive. If the user has more than one TU60 dual cassette unit, he should probably label the drives in consecutive order so that there will be no confusion when he is using the system.

The write protect tab on the System Cassette should usually be in the write-locked position so that data will not accidently be written on it; it is suggested that the user make a copy of this cassette as protection against loss or accidental destruction.

1.4 MOUNTING AND DISMOUNTING A CASSETTE

To mount a tape on a drive, hold the tape so that the open part of the cassette is to the left and the full reel is at the top. Set the top write protect tab to the desired position depending upon whether data is to be written on the tape.

Open the locking bar on the cassette drive by pushing it to the right away from the drive (see A in Figure 1-3). Next hold the tape up to the cassette drive at approximately a 45-degree angle and insert the tape into the drive by applying a leftward pressure while simultaneously rotating the cassette over the drive sprockets. This brings the tape into position against the read/write head. Push the tape into the unit so that when the cassette is properly mounted, the locking bar automatically closes over the cassette back edge. (Figure 1-3 illustrates this procedure.)

Press the rewind button on the cassette unit (see B in Figure 1-3; there is a rewind button for each drive). This causes the cassette to rewind to the beginning of its leader/trailer tape. (Pressing the rewind button a second time causes the cassette to rewind to the end of the leader/trailer tape and to the physical end-of-tape. The cassette unit will click; this sound is almost inaudible and the user may not hear it unless he is listening carefully. Normal usage requires that the user press the rewind button only once whenever he wishes to manually rewind a cassette). Even though tapes which are not actively being used on a drive should already be positioned at the beginning, the user should develop the habit of automatically rewinding a cassette. When the tape has finished winding, the cassette will stop moving. The cassette is now in place and ready for transfer operations.

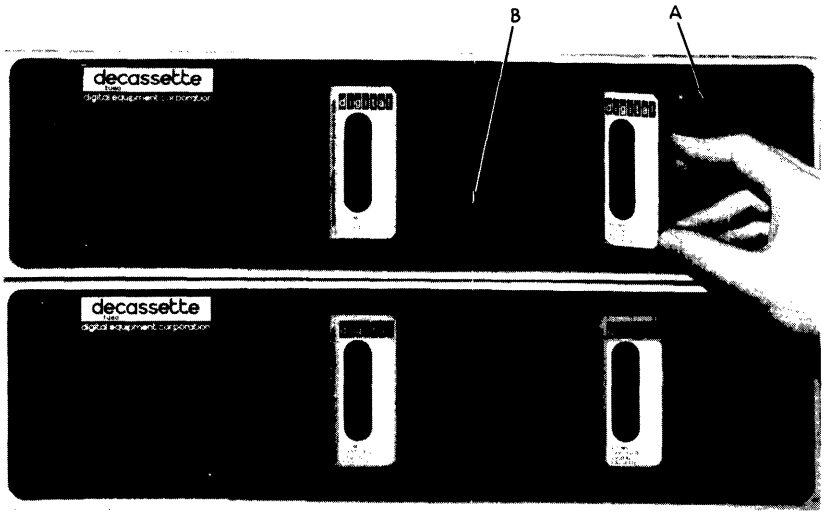


Figure 1-3 Mounting a Cassette

Before removing a cassette from a drive, the tape should always be rewound to its beginning. This can be done by pressing the rewind button on the cassette unit or by issuing the REwind Monitor Command as explained in Chapter 2. Rewinding a tape ensures that the clear leader/trailer tape will be the only tape exposed at the open part of the cassette. To remove a cassette from the cassette drive, open the locking bar and the cassette will pop out. When cassettes are not being actively used on a cassette drive, they can be stored in the small plastic boxes provided for this purpose by the manufacturer.

NOTE

> Before using a new cassette, or prior to using a cassette that has just been shipped or accidently dropped, mount the cassette on a drive so that the Digital label faces the inside of the unit and perform a rewind operation. Remove the cassette, turn it over, and perform another rewind operation. This packs the tape neatly in the cassette and places the full tape reel at the proper tension.

1.5 CONCERNING EXAMPLES

In the chapters that follow, care has been taken to include acutal machine printout whenever possible. In cases in which there may be some discrepancy as to whether a character was typed by the user or by the system, that typed by the system will be underlined.

CHAPTER 2
GETTING ON-LINE WITH THE CAPS-8 SYSTEM

2.1 SYSTEM PROGRAMS

The Cassette Programming System is distributed on a single cassette, called the System Cassette, which contains all the programs necessary for loading the Monitor into memory and creating and running system and user programs. The directory of the System Cassette is as follows:

```
C2BOOT.BIN  01/22/73  V1
MONTOR.BIN  01/22/73  V1
SYSCOP.BIN  01/25/73  V2
EDIT  .BIN  01/02/73  V1
PALC  .BIN  01/02/73  V1
BASIC .BIN  01/02/73  V1
```

System files are in binary format (see Section 2.2.1). The first two files on any System Cassette must be C2BOOT.BIN and MONTOR.BIN; these two files comprise a bootstrap and the Keyboard Monitor. C2BOOT.BIN loads the Monitor into memory from the System Cassette; the Keyboard Monitor links the user and the CAPS-8 System by providing a means of communication between the two. By accepting commands from the console terminal keyboard, the Cassette Keyboard Monitor allows the user to run system and user programs, save and recall files utilizing cassette storage, and create, assemble, and load programs.

2.2 SYSTEM CONVENTIONS

The following conventions concern file formats and file naming procedures and are standard for the CAPS-8 System, as well as for many other systems.

2.2.1 File Formats

The Cassette Programming System makes use of two types of file formats -- ASCII and Binary.

Files in ASCII format conform to the American National Standard Code for Information Interchange in which alphanumeric characters are represented by a 3-digit code. A table containing ASCII character codes in 7- and 8-bit octal is provided in Appendix A.

Binary format files consist of 12-bit binary words representing PDP-8 machine language code. The standard DEC binary format is used with the exception that no checksum is necessary. Binary files contain field addresses and memory instructions and are read directly into

memory for immediate execution. CAPS-8 System Programs are in binary format, and programs which the user assembles with PALC are translated into files in binary format.

2.2.2 Filenames and Extensions

System and user files are referenced symbolically by a name of as many as six alphabetic characters (A-Z) or digits (0-9), followed optionally by an extension of from 1 to 3 alphabetic characters or digits; (the first character in a filename must be alphabetic). The extension to a filename is generally used as an aid in remembering the CAPS-8 format of a file. Table 2-1 lists commonly accepted extensions -- the user may or may not conform to this list as he chooses; it is included here only as a guide:

Table 2-1 CAPS-8 Extension Names

Extension	Meaning
PAL	PALC source file (ASCII)
BIN	System or user binary format file
BAS	BASIC source file (ASCII)
TXT	Text file (ASCII)
DOC	Documentation file (ASCII)
DAT	Data file (ASCII or other)

Generally the user may call his files by any mnemonic filename and extension he chooses. In some cases, if he omits specifying an extension, the System Program he is running may assume an extension. For example, PALC assumes an extension of .PAL unless the user indicates another, and the Run command assumes .BIN unless another extension is specified.

2.2.3 Input/Output Devices

There are three available categories of input/output devices in the Cassette Programming System: console terminal keyboard (including paper tape reader and punch if an LT33 Teletype containing these units is used as the console terminal), cassette drives 0-7, and a line printer. There are no permanent device names in the CAPS-8 System. Command strings and I/O designations are entered in such a way that the user specifies the device by a drive number and the file by a filename; option characters allow the user to direct listings to the line printer or to otherwise change the normal operating procedure of a program. The System Cassette -- drive 0 -- is the default device if no drive number is specified. For example:

```
.DI/L
```

(DI is a Monitor command instructing the computer to print a directory listing of a cassette. Since no drive number is specified, drive 0 -- the System Cassette -- is assumed. The option character L sends output to the line printer instead of the console terminal, which is the normal output device.)

2.3 LOADING THE KEYBOARD MONITOR

The CAPS-8 hardware bootstrap and C2BOOT.BIN on the System Cassette are used to load the Cassette Keyboard Monitor into memory. (Both bootstraps are described in Appendix E.) Loading the Monitor is accomplished as follows:

1. Ensure that the computer and terminal are on-line.

2. Press and raise the HALT key. Make sure that the SINGLE STEP key is in a raised position.

3. Place the System Cassette (write-locked to protect data) onto cassette unit drive 0;

4. Press and raise the SW key.

At this point the RUN lamp should be on and the System Cassette should begin to move. The hardware bootstrap calls the first program on the System Cassette (C2BOOT.BIN) which in turn loads the Keyboard Monitor (MONTOR.BIN) into memory. If an error occurs during the loading process (for example, an error may be caused by the cassette being improperly mounted, by a missing file on the tape, or by the occurrence of an I/O error) no error message will inform the user since the Monitor is not completely in memory. Instead, the System Cassette may stop moving and the computer may loop or halt. If this is the case, steps 2-4 above should be repeated.

Once the Monitor has been loaded, the System Cassette stops moving and a dot is typed at the left margin of the console terminal page. This instructs the user that the Monitor is now in memory and ready to accept input commands.

2.4 USING THE KEYBOARD MONITOR

Each command to the Keyboard Monitor is typed at the console terminal keyboard in response to the dot at the left margin. A command is entered by pressing the RETURN key.

2.4.1 Making Corrections

Corrections may be made to the command line providing they are made before the line is entered (that is, before a carriage return has been typed). The RUBOUT key is used to correct typing errors. Pressing the

RUBOUT key once causes an open bracket ([) to be typed followed by the last character entered into memory. After this character is echoed on the console terminal it is deleted from memory. Successive RUBOUTs each cause one more character to be printed and deleted. The first non-RUBOUT character typed (after the last RUBOUT in a sequence) causes a closing bracket (]) to be printed, thus enclosing only the deleted portion of text within brackets. For example:

The user types: .R BACIC(RUBOUT)(RUBOUT)(RUBOUT)SIC
The console terminal shows: .R BACIC[CIC]SIC
The string is entered to the Keyboard Monitor as: .R BASIC

2.4.2 Special Characters (CTRL/C, CTRL/O, and CTRL/U)

Control can be returned to the Keyboard Monitor while under any of the System Programs by typing a CTRL/C (produced by holding down the CTRL key and simultaneously pressing the C key). If the Monitor is not still in memory, a CTRL/C causes a complete rebootstrap by reading the appropriate files from the System Cassette on drive 0. When it is ready to accept input, the Keyboard Monitor types a dot at the left margin of the teleprinter (i.e. console terminal) page.

Teleprinter output can be suppressed by typing a CTRL/O (produced by holding down the CTRL key and simultaneously pressing the O key). This allows execution of the program to continue, but stops all console printout. Typing a second CTRL/O will resume output again. Unless output is extremely lengthy, or unless the program is waiting for input from the user, processing of a program after an initial CTRL/O will usually be completed before the user is able to type a second CTRL/O. Printout will automatically resume when control is returned to the Keyboard Monitor (indicated by a dot at the left margin).

NOTE

CTRL/O does not prevent certain important error messages from printing on the console terminal.

A command line may be deleted completely, before it is entered, by typing a CTRL/U (produced by holding down the CTRL key and pressing the U key). This causes the current command line to be ignored and returns control to the Keyboard Monitor. The Monitor prints a dot at the left margin to indicate that it is ready to accept another command.

2.4.3 I/O Designations and Specification Options

Whenever the user runs a System Program or performs any I/O operation, he must indicate the file(s) to be accessed, the cassette drive(s) on which they are located, and any desired options associated with the operation. Procedures used in entering this information are explained below.

Monitor commands generally require only a single command line which specifies the unit drive number (in the range 0-7), filename(s), and option(s) in the following format:

```
.COMMAND DRIVE #:FILENAME.EXT/OPTION(S)
```

COMMAND represents one of the eight Monitor commands discussed in Section 2.5. The filename should be separated from the drive number by a colon. Options are alphabetic characters and are separated from the rest of the command line by the slash character (/). Successive options follow one another without any separating character. The command line is executed by typing a carriage return.

I/O specifications to System Programs follow a different format. First the System Program is called from the System Cassette using the Monitor Run command. The System Program then asks for the input filename, drive number, and options, and then the output filename, drive number, and options. This information is usually requested in two separate command lines, but the actual format varies between System Programs. Generally, the command strings appear as follows:

```
.R SYSTEM PROGRAM/OPTIONS
*INPUT-DRIVE #:FILENAME
*OUTPUT-DRIVE #:FILENAME
```

The appropriate chapter should be referenced for the accurate format.

Options are available in most System Programs and Monitor commands allowing the user to change the order or format of input and output operations from that which would normally be carried out by the program. Again, interpretation of options varies; the user should refer to the appropriate section or chapter to learn which options are available and what actions will result from their use.

2.5 KEYBOARD MONITOR COMMANDS

There are eight Keyboard Monitor commands available to the user. Commands are typed in response to the dot printed by the Monitor and are entered when the RETURN key is pressed. Each command consists of one or more alphabetic characters, followed by a space (or any non-alphabetic character). Any error made while utilizing these commands will result in a message informing the user (see Section 2.8). After occurrence of an error, control returns to the Keyboard Monitor and the command must be retyped. (Since several of the commands begin with the same letter, the user must be sure to note how much of the command must be entered in order to distinguish it from other commands. While it does not matter if too many characters are entered, too few will cause errors.)

2.5.1 Run Command

The Run command is of the form:

.R Drive #:Filename/Option(s)

The Run command instructs the Monitor to load and execute the file specified in the command line. The file should be in self-starting binary format (that is, the last location in the source file must be an origin setting which indicates the starting address of the file); as the file is not in self-starting binary format, the program will be loaded but execution will not begin; the user will have to proceed as though he were using the Load command (see Section 2.5.2). The user may omit specifying an extension as the Monitor assumes .BIN. For example:

```
.R CART.BIN
```

or

```
.R CART
```

Regardless of which command string the user types, the Monitor assumes .BIN, searches drive 0 for the file CART.BIN, and executes it.

Options allowed in the command line depend upon the program the user is running. Availability of options and results of their use are discussed in Chapters 3, 4, 5 and 6. No error occurs if the user specifies an option not allowed by a program; the option is simply ignored.

Multiple files may be executed using the Run command. Patches to programs, BASIC user-coded functions, and programs the user may have created using PALC can be executed as follows:

.R Drive #:PROG1,Drive #:PROG2,...PROGn/Option(s)

where n represents any number of programs as long as the total number of characters on the input specification line does not exceed 64. The user must enter programs in the command line in the order in which he wants them executed and must be careful to include appropriate starting, chaining, and return addresses (see Appendix E).

For example, assume the user has written a routine which will be used for debugging purposes; each time a certain condition is met during execution, this routine will be accessed, print a message and cause execution to halt. The routine has been created using the CAPS-8 EDITOR, assembled with PALC, and is stored on cassette drive 1 as DBG.BIN; it is loaded into memory with the user's program (TABLE.BIN stored on cassette drive 0). The programs are loaded as follows:

```
.R TABLE.BIN,1:DBG.BIN
```

Chapter 6, BASIC, contains an example of running multiple files in conjunction with the BASIC user-coded function feature.

2.5.2 Load Command

The Load command is used to load a binary file into memory and takes the form:

```
.L Drive #:Filename.ext/Option(s)
```

This command is similar to the Run command except that the computer halts after loading the file. To start execution, the user sets the correct starting address in the Switch Register, presses ADDR LOAD, CLEAR and CONT (if the file is in self-starting binary format, the user need only press CONT); appropriate addresses included in the program (see Appendix E) will return control to the Keyboard Monitor after execution.

Multiple files may be loaded in the same manner as in the Run command by simply specifying them in correct execution order on the command line:

```
.L Drive #:PROG1,Drive #:PROG2....PROGn/Option(s)
```

Again, n may represent any number of programs as long as the total number of characters on the command line does not exceed 64.

2.5.3 DAte Command

The DAte command is of the form:

```
.DA mm/dd/yy
```

where mm, dd, and yy represent the current month, day and year as entered by the user. (One or two-digit numbers in the range 0-99 are allowed in the DAte command. The Keyboard Monitor does not check for errors other than the entry of a number which is outside this range.) This date will then appear in directory listings (see Section 2.5.4), and the date of creation of all new files will be included. If the DAte command is not used, directory listings will contain only filenames, as illustrated in Section 2.1.

2.5.4 DIrectory Command

The DIrectory command is of the form:

```
.DI Drive #/Option(s)
```

and causes a directory listing of the cassette on the drive specified to be output on the console terminal. No colon is necessary after the

drive number. There are two options available for use with the DIrectory command:

Table 2-2 Directory Options

Option	Meaning
/L	Causes the listing to be output on the line printer rather than the console terminal.
/F	Causes a "fast" listing to be produced (omitting creation dates and version numbers).

In the following example a directory of cassette drive 2 is requested and output (the version number in the directory listing reflects the number of times the file has been accessed and changed using the CAPS-8 EDITOR; see Chapter 3, Section 3.2.3):

```
10/29/72
FILE   .BIN  03/17/72
ABCDEF.PAL
A      .ASC            V3
B      .               V22
```

This same directory using the F option will be reduced to:

```
10/29/72
FILE   .BIN
ABCDEF.PAL
A      .ASC
B      .
```

2.5.5 DElete Command

The DElete command is of the form:

.DE Drive #:Filename.ext

and causes the filename on the specified drive number to be deleted from the directory. The filename is replaced by the name *EMPTY in the directory listing and the file can no longer be referenced. Only one file may be specified in the DElete command string at a time.

For example, assume the user wishes to delete the filename MATH.DAT from the directory of cassette drive 3. He types:

```
.DE 3:MATH.DAT
```

and then obtains a directory listing of drive 3. The directory will appear as follows:

```
11/17/72
TAPE   .BAS   11/02/72
*EMPTY.
TOR    .ASC   11/07/72  V3
```

where *EMPTY represents the deleted filename MATH.DAT.

2.5.6 Zero Command

The Zero command is of the form:

.Z Drive #:Filename

and specifies that the sentinel file of the indicated cassette is to be moved so that it immediately follows the file indicated in the command line. (See Chapter 1 for a description of the sentinel file.) All files following the sentinel file are deleted from the cassette and that portion of the tape is completely reusable.

For example, assume cassette drive 3 contains the following directory:

```
LOOK   .ASC   10/23/72  V2
BASE   .BAS
FOUR   .BIN   11/17/72
*EMPTY.
RACE   .E
```

and the user wishes to save only the first three files. He uses the Zero command as follows:

```
.Z 3:FOUR.BIN
```

and the sentinel file is placed immediately after the file FOUR.BIN. The directory now reads:

```
LOOK.ASC   10/23/72  V2
BASE.BAS
FOUR.BIN   11/17/72
```

When no filename is specifed in the command line, for example:

```
.Z 1
```

the cassette is said to be zeroed, or completely deleted of files; the sentinel file is moved to the beginning of the cassette so that the entire tape is available for use. This method is useful in "cleaning up" cassettes which may contain several *EMPTY files in the directory listing but have become full and therefore unavailable for further use. First, any needed files are transfered to another cassette using SYSCOP (see Chapter 4), then the directory of the old cassette is zeroed. The sentinel file is written at the beginning of the tape making the cassette completely reusable.

All new tapes must be zeroed before they are first used. This ensures that a sentinel file is present on the tape and moved to the beginning of the tape.

2.5.7 REwind Command

The REwind command is of the form:

```
.RE Drive #
```

and causes the cassette on the drive number specified to be rewound to its beginning. (The user can also cause the tape to rewind by pressing the rewind button on the cassette unit.) System Programs and Monitor commands always rewind a cassette before accessing a file, but if the user delevopes the habit of rewinding the cassette himself he performs a timesaving action. A cassette should always be rewound before it is removed from a drive.

2.5.8 Version Command

The Version command is used to find out the version number of the Monitor currently in use. Typing:

```
.V
```

instructs the Monitor to respond with the appropriate number. For example:

```
.V
V1.2
```

indicates that version 1.2 is currently in use.

2.6 NOTES ON DEVICE HANDLERS

Device handlers for the CAPS-8 System are described in Appendix E. A few notes of interest concerning their use are included here.

The line printer performs a form feed operation before beginning an output task. Characters are unpacked from the output buffer and printed. A form feed is also produced following the completion of an output task. The line printer handler is capable of handling only an 80 column printer.

If the console terminal is an LT33 Teletype containing reader and punch units, these may be used as input/output devices in conjunction with the Teletype keyboard. To punch a tape, simply place the punch unit to ON; to read a tape, place the reader unit to START. Characters will be printed on the Teletype keyboard as they are read or punched. Binary tapes may not be punched.

NOTE

> The purpose of the Cassette Programming System is the elimination of paper tape procedures. Cassettes provide a more convenient, reliable and faster means of program storage than paper tape. Therefore, although paper tapes may be read and punched using the LT33 paper tape units, there is no support for this type of I/O and its use is not encouraged.

If the user's program does not over-write certain areas of memory, the parts of the Monitor which are in these locations are available for use. This allows the user who takes advantage of writing his own programs in the PAL machine language to access system handlers and to restart or rebootstrap the Cassette Keyboard Monitor after program execution. Information concerning Monitor Service Routines, I/O routines, device handlers, and internal descriptions of the Keyboard Monitor are provided in Appendix E.

2.7 KEYBOARD MONITOR ERROR MESSAGES

The following error messages may occur when the Keyboard Monitor is used incorrectly:

Table 2-3 Keyboard Monitor Error Messages

Message	Meaning
BAD COMMAND	The user has failed to follow the correct syntax for Monitor commands. This may be the result of mispelled commands or too many or improper arguments in a command string.
FILE NOT FOUND	The Monitor could not locate the file (or files) specified. The user should check to be sure that filenames are spelled correctly and that the unit drive number specified is correct.
INPUT ERROR ON UNIT n OUTPUT ERROR ON UNIT n	An I/O error has occurred on the cassette drive specified. This may be caused by an incorrectly formatted cassette or may be due to a timing error. The user should try the I/O transfer using another cassette.
UNIT n NOT READY	There is no cassette on the drive specified, or no such drive exists.
UNLOCK UNIT n	The user tried to write data when the write protect tab of the cassette on the drive specified was write-locked. To write data this tab must be write-enabled.

CHAPTER 3
SYMBOLIC EDITOR

3.1 INTRODUCTION

The CAPS-8 Symbolic EDITOR is used to create and modify ASCII source files so that these files may be used as input to other System Programs such as BASIC and PALC.

The EDITOR considers a file to be divided into logical units called pages. A page of text is generally 50-60 lines long, and corresponds approximately to a physical page of a program listing. (Note that this is not the same as a memory page). The EDITOR reads one page of text at a time from the input file into its internal buffer where the page becomes available for editing. The Editor contains commands for creating, modifying, or deleting characters, lines, or complete logical pages of text. All commands consist of a single letter or a letter with arguments, and are executed by typing the RETURN key.

3.2 CALLING AND USING THE EDITOR

To call the EDITOR from the System Cassette, type:

```
.R EDIT/Options
```

in response to the dot (.) printed by the Keyboard Monitor.

3.2.1 EDITOR Options

There are two options available for use with the EDITOR; these are described in Table 3-1. (Option usage has been previously discussed in Chapter 2, Section 2.4.3).

Table 3-1 EDITOR Options

Option	Meaning
/B	Convert two or more spaces to a TAB when reading from input device.
/M	More than one file will be used for input. (When one of these commands--E, F, J, N, R, or Y--is issued and an end-of-file is encountered, the EDITOR pauses and requests that the user specify another input file, thus allowing continuation of the command. If the /M option has not been previously specified in the input line, the end-of-file condition remains in effect. See Section 3.9 for an example.)

3.2.2 Input and Output Specifications

After the EDITOR has been called from the System Cassette it asks for the input specification as follows:

```
*INPUT FILE-
```

The user responds with the input cassette drive number and the input filename and extension, if any. For example:

```
*INPUT FILE-1:ABA.PAL
```

If only a filename (and no input cassette drive) is specified, the default device--drive 0--is assumed; the EDITOR prints the user's input specification line, only first it includes the assumed default device before echoing the filename, as illustrated below:

```
*INPUT FILE-0:AB
```

The user has typed the filename AB, but before this is printed, the EDITOR inserts 0: and then goes on to echo AB. If the input file is not found or if a syntax error occurs, the EDITOR prints a question mark (?), types an asterisk (*) at the left margin, and waits for another input designation. Any number of input files is permitted.

If no input specification is made, (that is, a carriage return only has been typed in response to the INPUT request), a new file will be created using the console terminal keyboard as the input device. The EDITOR allows input from the keyboard via the Append command (see Example Using the EDITOR for an illustration of this method of creating a program).

If more than one input file is to be entered, the /M option must be specified when the EDITOR is called from the System Cassette. The user responds to the INPUT FILE line with the drive number and filename of the first input file. He enters output information as described next, and then edits his file. When the end-of-file is reached during the editing procedure, the EDITOR again prints the INPUT FILE request and the user responds with the drive number and filename of the second file. When the user finishes editing his final file and no more input files are available, he responds to the EDITOR's INPUT request by typing a carriage return; the EDITOR continues and closes the output file. All input files are combined under the one filename specified in the output line.

The EDITOR initially requests output information by printing:

```
*OUTPUT FILE-
```

The user responds with the output drive number and filename. For example :

```
*OUTPUT FILE-2:OUT.EXT
```

Again, if no device is designated, drive 0 is assumed and echoed.

If the output file is to have the same name as the input file, the user need only type the correct output drive number followed by a carriage return; the EDITOR will echo the assumed name. For example:

```
*INPUT FILE-1:FILE.BAS
*OUTPUT FILE-2:FILE.BAS
```

The EDITOR allows only one output file and creates the header for this file on the specified cassette, deleting any file already on that cassette under the same name (and replacing it with *EMPTY in the directory listing) and leaving the cassette correctly postioned for further output.

NOTE

> If no output designation is specified (that is, a carriage return only has been typed in response to the OUTPUT FILE request), the only output operations which may be performed are L (list buffer on the console terminal) or V (list buffer on the line printer).

Only cassette files in ASCII mode are acceptable for use by the EDITOR. No error message is given if non-ASCII files are input, but the results of editing operations are garbled.

Once I/O file designations are entered, the Symbolic EDITOR is ready to accept commands from the keyboard. It signifies this by printing a number sign (#) at the left margin; this symbol occurs whenever the EDITOR is waiting for a command.

3.2.3 Version Numbers

Each time a filename is indicated in response to the output file specification line, the number 0 is assigned to it. This number (called the version number) signifies that a new file has been created and that it has not been previously edited or referenced under this filename.

The user may call a file from a specified cassette, make corrections to it and change it any number of times before he is finally satisfied with it or ready to use it for some other operation. In this case, he may reference the file in the output specification line by specifying only the output cassette drive number followed by a carriage return, since the filename itself will not be changed. Each time he does

this, the version number of the file is increased by 1. When the version number of a file has been incremented in this manner so that it is greater than 0, it appears in directory listings on the same line as the filename (see Chapter 2).

NOTE

Version numbers associated with edited files should not be confused with the V Monitor command, which prints the version of the Monitor currently in use.

3.3 MODES OF OPERATION

The EDITOR operates in one of two different modes: command mode or text mode. In command mode all input typed on the keyboard is interpreted as commands instructing the EDITOR to perform some operation. In text mode, all typed input is interpreted as text to replace, be inserted into, or be appended to the contents of the text buffer.

3.3.1 Transition Between Modes

Immediately after being loaded into memory and started, the EDITOR is in command mode. The special character # is printed at the left margin of the teleprinter page indicating that the EDITOR is waiting for a command. All commands are terminated by pressing the RETURN key.

In text mode, the EDITOR performs I/O operations on text stored within the text buffer. Text is input to the EDITOR buffer until a form feed is encountered. A line of text is terminated by a carriage return. If no carriage return is present, the text entered on the current line is ignored. The buffer has room for approximately 5200 (decimal) characters. When text has been input to the extent that there are only 256 decimal locations available in the buffer, the console terminal rings a warning bell. From this point on, whenever a carriage return is detected during text input, control returns to the EDITOR command mode and the bell is rung. This line-at-a-time input may continue until the absolute end-of-buffer is encountered. At this point, no more text will be accomodated in the buffer; a "?" is printed and control returns to command mode every time the user attempts to input more text.

3.4 SPECIAL CHARACTERS AND FUNCTIONS

A number of the console terminal keys have special operating functions. These keys and their associated functions are described below.

3.4.1 RETURN Key

In both command and text modes, typing the RETURN key causes a carriage return and line feed operation and signals the EDITOR to process the information just typed. In command mode, it allows the EDITOR to execute the command just typed. A command will not be executed until it is terminated by the RETURN key (with the exception of =, explained later). In text mode, RETURN causes the line of text which it follows to be entered in the text buffer. A typed line is not actually part of the buffer until terminated by the RETURN key.

3.4.2 Erase (CTRL/U)

The erase character (CTRL/U combination) is used for error recoveries in both command and text modes. It is generated by holding the CTRL key while simultaneously typing the U key. When used in text mode, CTRL/U cancels everything to its left back to the beginning of the line; the EDITOR echoes ↑U and performs a carriage return/line feed (CR/LF); the user then continues typing on the next line. When used in command mode, CTRL/U cancels the entire command; the EDITOR performs a CR/LF and prints a #. The erase character cannot cancel past a CR/LF in either command or text mode.

3.4.3 RUBOUT Key

Rubout is used in error recovery in both command and text modes. In text mode typing the RUBOUT key echoes a backslash (\) and deletes the last typed character. Repeated rubouts delete from right to left up to, but not including, the CR/LF which separates the current line from the previous one. For example:

```
THE QUJICK\\\\ICK BROWN FOX
```

will be entered in the buffer as:

```
THE QUICK BROWN FOX
```

When used in command mode, RUBOUT is equivalent to the CTRL/U feature and cancels the entire command; the EDITOR prints a #, performs a CR/LF, and waits for another command to be entered.

3.4.4 Form Feed (CTRL/FORM)

A form feed signals the EDITOR to return to command mode. A form feed character is generated by typing the CTRL and FORM keys simultaneously. This combination is typed while in text mode to indicate that the desired text has been entered and that the EDITOR should now return to command mode. The EDITOR performs a CR/LF and

prints a # in response to a CTRL/FORM to indicate that it is back in command mode. CTRL/G is usually equivalent to CTRL/FORM except in the case of a SEARCH command, as explained in Section 3.8.1.

3.4.5 The Current Line Counter (.)

The EDITOR keeps track of the implicit decimal number of the line on which it is currently operating. The dot (produced by typing the period key) stands for this number and may be used as an argument to a command. For example, .L means list the current line; .-1,.+1L means list the line preceding the current line, the current line, and the line following it, then update the dot (current line counter) to the decimal number of the last line printed.

The following commands affect the current line counter as indicated:

1. After a Read or Append command, the current line counter is equal to the number of the last line in the buffer.

2. After an Insert or Change command, the current line counter is equal to the number of the last line entered.

3. After a List or Search command, the current line counter is equal to the number of the last line listed.

4. After a Delete command, the current line counter is equal to the number of the line immediately after the deletion.

5. After a Kill command, the current line counter is equal to 0.

6. After a Get command, the current line counter is equal to the number of the line printed by the GET.

7. After a Move command, the current line counter is not updated and remains whatever it was before the command was issued.

3.4.6 Slash (/)

The slash symbol (/) has a value equal to the decimal number of the highest numbered line in the buffer. It may also be used as an argument to a command. For example: 10,/L means list from line 10 to the end of the buffer.

3.4.7 LINE FEED Key

Typing the LINE FEED while in command mode is equivalent to typing .+1 and will cause the EDITOR to print the line following the current one and to increment the value of the current line counter by one.

3.4.8 ALT MODE Key

Typing the ALT MODE key while in command mode will cause the line following the current line to be printed and the current line counter to be incremented by one. If the current line is also the last line in the buffer, typing either ALT MODE or LINE FEED will gain a response of ? from the EDITOR indicating that there is no next line. Some console terminals provide an escape key (ESC) in place of the ALT MODE. Their functions are identical.

3.4.9 Right Angle Bracket (>)

Typing the right angle bracket (>) while in command mode is equivalent to typing .+1L and will cause the EDITOR to echo > and then print the line following the current line. The value of the current line counter is increased by one so that it refers to the last line printed.

3.4.10 Left Angle Bracket (<)

Typing the left angle bracket (<) while in command mode is equivalent to typing .-1L and will cause the EDITOR to echo < and then print the line preceding the current line. The value of the current line counter is decreased by one so that it refers to the last line printed.

3.4.11 Equal Sign (=)

The equal sign is used in conjunction with the pointer's dot (.) or slash (/). When typed in command mode the equal sign causes the EDITOR to print the decimal value of the argument preceding it. In this way the user may determine the number of the current line (.=), or the total number of lines in the buffer (/=), or the number of some particular line (/-8=) without counting lines from the beginning of the buffer. No carriage return need be typed following the equal sign.

3.4.12 Colon (:)

Typing a colon produces the same result as the equal sign (=).

3.4.13 Tabulation (CTRL/TAB)

The EDITOR is written in such a way as to simulate tab stops at 8-space intervals across the teleprinter page. When the CTRL key is held down and the TAB key is typed, the EDITOR produces a tabulation. A tabulation consists of from one to eight spaces, depending on the number needed to bring the carriage to the next tab stop. Thus, the

EDITOR may be used to produce neat columns on the teleprinter or line printer page. The tab function is used in conjuction with the /B option (for input and output) to allow the user to produce and control tabulations in the text buffer during input operations. On input (under a Read command), the EDITOR will replace a group of two or more spaces with a tabulation if the user has specified the /B option.

3.5 COMMAND STRUCTURE

A command directs the EDITOR to perform a desired operation. Each command consists of a single letter, preceded by zero, one, two or three arguments. The command letter tells the EDITOR what operation to perform; the arguments usually specify which numbered line or lines of text are affected. Command format is illustrated in Table 3-2, where E represents any command letter.

Table 3-2 Command Format

Type of Command	Command Format	Meaning
No Argument:	E	Perform operation E
One argument:	nE	Perform operation E on the referenced line.
Two Arguments:	m,nE	Perform operation E on lines m through n, inclusive.
Three Arguments:	m,n$jE	This combination is used by the MOVE command only and is explained in Section 3.6.3.

3.6 COMMAND REPERTOIRE

Commands to the EDITOR are grouped under three general headings:

Input Commands
Output Commands
Editing Commands

Explanation of the three types of commands is detailed in the following sections. Each command description will state if the EDITOR returns to command mode after completing the operation specified by the command. All commands are entered when the RETURN key is typed.

The EDITOR prints an error message consisting of a question mark whenever the user has requested nonexistent information or used inconsistent or incorrect format in typing a command. For example, if a command requires two arguments, and only one (or none) is provided, the EDITOR will print ?, perform a carriage return/line feed, and ignore the command as typed. Similarly, if a nonexistent command character is typed, the error message ? will be printed, followed by a carriage return/line feed; the command will be ignored. However, if

an argument is provided for a commmand that does not require one, the argument will be ignored and the normal function of the commmand performed. For example:

User Types:	Result:
L ?	The buffer is empty. The user is asking for nonexistent information.
7,5L ?	The arguments are in the wrong order. The EDITOR cannot list backwards.
17$10M ?	This command requires two arguments before the $; only one was provided.
H ?	The user types a nonexistent command letter.

3.6.1 Input Commands

Two commands are available for inputting text, and are described in Table 3-3.

Table 3-3 Input Commands

Command	Format	Action and Explanation
A	#A	Append the incoming text from the console terminal keyboard to the information already in the buffer (if there is no input file the buffer will be empty initially). The EDITOR will enter text mode upon receiving this command and the user may then type in any number of lines of text. The new text will be appended to the information already in the buffer, if any, until a form feed (CTRL/FORM key combination) is typed; control then returns to command mode. By using the Append command with an empty buffer, a symbolic program may effectively be generated on-line by entering the program via the keyboard. Any rubout encountered during execution of an Append command will delete the last typed character. Repeated rubouts will delete from right to left up to but not beyond the beginning of the current line.

Table 3-3 Input Commands (Cont'd)

Command	Format	Action and Explanation
R	#R	Read a page of text from the input file on the specified unit drive. The EDITOR will read information from the input file until a form feed character (CTRL/FORM key combination) is detected or until the EDITOR senses a text buffer full condition. All incoming text except the form feed is appended to the contents of the text buffer. Information already in the buffer remains there. NOTE In both these commands, the EDITOR ignores ASCII codes 340 through 376. These codes include the codes for the lower case alphabet (ASCII 341-372). The EDITOR returns to command mode only after the detection of a form feed or when a buffer full condition is reached.

3.6.2 Output Commands

Output commands are subdivided into list and text transfer commands. List commands will cause the printout of all or any part of the contents of the text buffer to permit examination of the text. Text transfer commands provide for the output of form feeds, corrected text, or for the duplication of pages of an input file. List or text transfer commands do not affect the contents of the buffer.

List Commands

The commands in Table 3-4 cause part or all of the contents of the text buffer to be listed on the console terminal or line printer.

Table 3-4 List Commands

Command	Format	Action and Explanation
L	#L	LIST the entire page. This causes the EDITOR to list the entire contents of the text buffer on the console terminal.
L	#nL	LIST line n. This line will be printed followed by a carriage return and a line feed.

Table 3-4 List Commands (Cont'd)

Command	Format	Action and Explanation
L	#m,nL	LIST lines m through n inclusive on the console terminal.
V	#V	List the entire text buffer on the line printer (if one is available).
V	#nV	List line n of the buffer on the line printer.
V	#m,nV	List line m through n inclusive on the line printer.

The EDITOR remains in command mode after a list command, and the value of the current line counter is updated so as to equal the number of the last line printed.

Text Transfer Commands

The following commands control the output of text and form feeds. The EDITOR is designed to minimize the possibility of illegal or meaningless characters being written into a source file; therefore the illegal (nonexistent) codes 340-376 and 140-177, and most illegal control characters will not be output.

Table 3-5 Text Transfer Commands

Command	Format	Action and Explanation
E	#E	Output the current buffer to the output file and transfer all input to the output file; close the output file.
P	#P	Transfer the entire contents of the text buffer to the output buffer.
P	#nP	Transfer line n only to the output buffer.
P	#m,nP	Transfer lines m through n inclusive (where m must be less than n) to the output buffer. When the output buffer becomes full, the text is output to the indicated output file. The P command automatically outputs a FORM character (214) after the last line of output.
N	#N	Transfer the contents of the text buffer to the output buffer, delete the text buffer and read in the next logical page of text from the input file.

Table 3-5 Text Transfer Commands (Cont'd)

Command	Format	Action and Explanation
N	#nN	Execute the above sequence n times. If n is greater than the number of pages of input text, the command will proceed in the specified sequence until it reads the end of the input file, then it will return to command mode. The N command cannot be used with an empty text buffer. A ? is printed if this is attempted.
Q	#Q	Immediate end-of-file. The Q command causes the entire text buffer to be output. All text written into the output buffer is then written into the output file and the file closed, with control returning to the Cassette Keyboard Monitor.

3.6.3 Editing Commands

The following commands permit deletion, alteration, or expansion of text in the buffer.

Table 3-6 Editing Commands

Command	Format	Action and Explanation
B	#B	List the number of available memory locations in the text buffer. The EDITOR returns the number of locations on the next line. To estimate the number of characters that can be accomodated in this area, multiply the number of free locations by 1.7.
C	#nC	Change line n. Line n is deleted, and the EDITOR enters text mode to accept input. The user may now type in as many lines of text as he desires in place of the deleted line. If more than one line is inserted, all subsequent lines will be automatically renumbered and the line count will be updated appropriately. A CTRL/FORM terminates the command.
C	#m,nC	Change lines m through n inclusive (m must be numbered less the n). Lines m through n are deleted and the EDITOR enters text mode allowing the user to type in any number of lines in their place. All subsequent lines will be automatically

Table 3-6 Editing Commands (Cont'd)

Command	Format	Action and Explanation
		renumbered to account for the change and the line count will be updated. After any Change operation, a return to command mode is accomplished by typing a CTRL/FORM. After a Change, the value of the current line counter (.) is equal to the number of the last line input. The C command utilizes the Text Collector in altering text (see Section 3.7).
D	#nD	Delete line n. Line n is removed from the text buffer. The current line counter and the numbers of all succeeding lines are reduced by one.
D	#m,nD	Delete lines m through n inclusive. The space used by the line to be deleted is reclaimed as part of the Delete function (refer to Section 3.7, Text Collection).
F	#F	Used during a string search. Find the next occurrence of the string currently being searched for (see Section 3.8.2, Inter-Buffer Character String Search).
G	#G	Get and list the next line which has a label associated with it. (A label in this context is any line of text which does not begin with a space, slash, TAB, or RETURN). The EDITOR begins with the line following the current line (line .+1) and tests for a line with a label. This will most often be a line beginning with a tag; it might also be a line containing an origin. For example: TAD (This is the current line) DCA /THIS IS A COMMENT HERE, 0 (This line would be printed by the command G) TAD ISZ *5000 (This line would also be printed if another G were typed)
G	#nG	Get and list the next line which begins with a label; the EDITOR begins at line n and tests it and each succeeding line as described in the preceding example.

Table 3-6 Editing Commands (Cont'd)

Command	Format	Action and Explanation
		Both G and nG update the current line counter after finding the specified line. However, if either version of the GET command reaches the end of the buffer before finding a line beginning with an ASCII character other than a tab, slash, or space, the current line counter retains the value it was assigned before the GET was issued, and a ? is typed to indicate that no tagged line was found. The EDITOR remains in command mode after a GET command.
I	#nI	Insert the typed text before line n until a form feed (CTRL/FORM) is encountered. The EDITOR enters text mode to accept input. The first line typed becomes the new line n. Rubouts are recognized. Both the line count and the numbers of all lines following the insertion are increased by the number of lines inserted. The value of the current line counter is equal to the number of the last line inserted. To reenter command mode, the CTRL/FORM key combination must be typed (terminating text mode). If this is not done, all subsequent commands will be interpreted erroneonusly as text and entered in the program immediately after the insertion.
I	#I	Insert text before line 1 (when used without an argument).
J	#J	Initiate an inter-buffer string search (See Section 3.8.2, Inter-buffer Character String Search).
K	#K	Kill (delete) the entire page in the buffer. The values of the special characters (/) and (.) are set to zero. The EDITOR remains in command mode.
		NOTE The EDITOR ignores the commands nK or m,nK. This prevents the buffer from accidently being destroyed if the user intended to type a List command (m,nL).

Table 3-6 Editing Commands (Cont'd)

Command	Format	Action and Explanation
M	#m,n$jM	Move lines m through n inclusive to before line j (m must be numerically less than n and j may not be in the range between m and n). Lines m through n are deleted from their current position and are inserted before line j. The lines are renumbered after the move is completed although the value of the current line counter (,) is unchanged, as moving lines does not use any additional buffer space. (The $ character is produced by typing a SHIFT/4.) A line or group of lines may be moved to the end of the buffer by specifying j as /+1. For example, 1,10$/+1M. Since the MOVE command requires three arguments, it must have three arguments in order to move even one line. This is done by specifying the same line number twice. For example, 5,5$23M. This will move line 5 to before line 23. The EDITOR remains in command mode after a Move command.
S	#nS	Search line n for the character specified after typing the S and a carriage return. Allow modification of the line when the character is found. (See Section 3.8.1, Single Character String Search.)
Y	#nY	Skip to a logical page in the input file, without writing any output. For example, #5Y. This command reads through 4 logical pages of input, deleting them without producing output. The fifth page is read into the text buffer and control automatically returns to command mode. If there are no more pages of input, the EDITOR issues a ? and returns to command mode.
$	#$TEXT" #$TEXT' #"	Perform a character string search for the string TEXT (see Section 3.8.2, Intra-Buffer Character String Search). Following a string search, #" causes a search for the next occurrence of the string.

3.7 TEXT COLLECTION

The CAPS-8 EDITOR contains an automatic text collector which reclaims buffer space following the use of a D, S, or C command. If a full

buffer condition is reached, the user may output lines of text (using the P command, for example), and then delete these lines from the buffer--text collection is automatic and always occurs on the three commands mentioned above.

NOTE

If extremely large amounts of text are deleted, the text collection process could take several seconds. For small amounts of text, no appreciable time is lost.

3.8 CHARACTER SEARCHES

Two types of searches were mentioned in Table 3-6--the standard character search and the character string search. Each is explained in turn.

3.8.1 Single Character Search

The single character search may take one of the following forms:

```
#S
or
#nS
or
#m,nS
```

where m and n represent line numbers (m<n), and S initiates the search command. This command searches the entire text buffer (or the line(s) indicated) for the search character. The search character is typed by the user after he types the RETURN key which enters the command, and does not echo on the teleprinter. The EDITOR prints the contents of the entire buffer or the indicated line(s) until the search character is found. When the search character is found, printing stops and the user types a response chosen from the following table:

Table 3-7 Search Character Options

Option	Result
text	Enter text at that point at which the search character was found and printing stopped.
CTRL/G (bell rings)	Change the search character to the next character typed; search continues. If the character is not contained in the line, the remainder of the line will be typed and control will be returned to command mode. (For example, CTRL/G CTRL/G would cause the remainder of the line to be listed.)

Table 3-7 Search Character Options (Cont'd)

Option	Result
CTRL/FORM	Continue searching for the next occurrence of the character.
RETURN key	End line here, deleting all subsequent text on that line.
LINE FEED key	Make two lines out of the current line. Typing a line feed actually inserts a carriage return without returning control to command mode.
RUBOUT key	Delete characters from the line. A rubout echoes a backslash (\) for each character deleted. When all characters have been deleted, echoing of "\" stops.

3.8.2 Character String Search

The character string search can identify a given line in the buffer by the contents of that line or any unique combination of characters. This search returns the line number as a parameter that can be used to further edit the text. There are two types of string search available: intra-buffer search and inter-buffer search.

Intra-Buffer Character String Search

The intra-buffer search scans all text in the current buffer for a specified character string. If the string is not found, a ? is printed and control returns to command mode. If the string is found, the number of the line which contains the string is put into the current line counter and control waits for the user to issue a command. Thus, searching for a character string in this manner furnishes a line number which can then be used in conjunction with other EDITOR commands. This provides a useful framework for editing, as it eliminates the need to count lines or search for line numbers by listing lines.

An intra-buffer search is signalled by typing the ALT MODE key (which echoes as $) in response to the # printed by the EDITOR. The user then types the string to be found (as many as 20 characters may be specified--any additional characters typed are echoed but not included in the search). The search string cannot be broken across line boundaries. Typing a single quote (') terminates the character string; when the RETURN key is typed the search is performed beginning at line 1 of the text buffer. Use of the double quote (") causes the search to begin at the current line +1. (Use of ' and " as command elements prohibits their use in the search string. An incorrect response resets the current line counter to the beginning of the buffer.)

For example, assume the text buffer contains the following text:

```
ABC DEF GJO
1A2B3C4D5E6
.STRINGABCD
.
.
```

The user wants to list the line that contains ABC; this could be done by typing:

```
#$ABC'L
```

The search begins with line 1 and continues until the string is found. The current line counter is set equal to the line in which the string ABC occurred, and the L command causes the line to be printed as follows:

```
ABC DEF GJO
```

Control returns to command mode, awaiting further commands. If the user wanted to find the next reference to ABC, he could type:

```
#"L
```

In this case, " is a command which causes the last string searched for to be used again, with the search beginning at the current line +1. It is not necessary to enter the search string again. The command may be used several times in succession. For example, if the user wanted to find the fourth occurrence of a string containing the characters FEWMET he could type:

```
#$ FEWMET'''''L
```

This command lists the line which contains the fourth occurrence of that string. The L (List) command (or any other command code) can be given following either ' or ". The L command causes the line to be listed when and if it is found.

In order to clear the text string buffer, the user can type:

```
#$'
```

The system responds with a question mark and the text string buffer is cleared.

The properties of the commands ' and " allow for easy and useful editing, as the following example illustrates. In order to change CIF 20 to CIF 10, the user can issue the following commands:

```
#$DUM,'$CIF 20"C
CIF 10  /NEW FIELD (CTRL/FORM)
```

The above set of instructions first causes the EDITOR to start at line 1 and search for a line beginning with DUM,. A search is then made for CIF 20, starting from the line after the line containing the string DUM, . When this string is found, the line number of the line containing the string CIF 20 becomes the current line number. The C command is given, and the user then changes the line to the correct instruction, CIF 10 /NEW FIELD .

Since this search feature produces a line number as a result, any operations which can be done by explicitly specifying a line number can be done by specifying a string instead. For example:

```
#$STRING'+4L
```

will list the fourth line after the first occurrence of the text STRING in the text buffer.

```
#$LABEL1,',LABEL2,"L
```

will list all lines between the two labels, inclusive.

```
#$PFLUG'S
```

will do a character search on the line which contains PFLUG. (The user types the search character after typing the RETURN key that enters the line.)

In cases where both strings and explicit numbers are used, strings should be used first. For example, the following commands:

```
#1+$BAD!'L
```

will not list the next line after the string BAD! occurs. The correct syntax is:

```
#$BAD!'+1L
```

Inter-Buffer Character String Search

The inter-buffer search scans the current text buffer for a character string. If the string is not found, the current buffer is written to the output file, the buffer is cleared, and the next buffer is read from the input device. The search then resumes at line 1 of the new buffer. This process continues until either the string is found or no more input is left. If input is exhausted, control returns to command mode with all the text having been written to the output file. If the string is found, control returns to command mode with the current line equal to the number of the line containing the first occurrence of the string. For example, a command to find the character string GONZO may

appear as follows:

```
#J
$GONZO'
#.=0024
```

The J command initiates an inter-buffer search; the $ is printed automatically by the EDITOR, and the user types in the character string he wishes to search for. The search proceeds, and when the string is found, control returns to command mode. The user types the .= construction to discover the number of the line in the current buffer on which the string is contained. To find further occurrences of the string GONZO, the user can use the F command. The F command uses the last character string entered to search the buffer starting from the current line count +1.

```
#F
#.=0106
```

The above example causes a search for the string GONZO starting at the current line +1. If no output file is specified in the J or F commands, the EDITOR reads the next input buffer without attempting to produce any output. This provides an easy way of paging through text for a particular string.

After the J or F commands have processed the entire input file, an E or Q command must be executed to close the output file.

The following two commands may be used to abort the string search command, once given:

Table 3-8 Terminating a String Search

Command	Explanation

CTRL/U	A CTRL/U will return control to the EDITOR command mode if executed while entering text in a string search command; the string search command is ignored, as in the following example:
	
	`#J`
	`$WORD↑U`
	
	`#`
	
	The inter-buffer search for the characters WORD was aborted by the user typing ↑U before terminating the string with ' or ".

RUBOUT	Executing the RUBOUT key while entering text for use in a string search causes the text so far entered to be ignored and allows a new string to be inserted. The EDITOR answers the command by typing

Table 3-8 Terminating a String Search (Cont'd)

Command	Explanation
	$, as seen in the following example: #$CHAR (RUBOUT) $

An example of the use of the character string search is contained in the EDITOR Demonstration Run found at the end of this chapter.

3.9 EDITOR ERROR MESSAGES

Errors made by the user while running the EDITOR may be of two types. Minor errors (such as an EDITOR command string error, an attempt to execute a read or write command without assigning a device, or a search for a nonexistent string) will cause a question mark to be typed at the left margin of the teleprinter paper. The command may be retyped.

Major errors force control to return to the Keyboard Monitor and may be due to one of the causes listed in Table 3-9. These errors cause a message to be typed in the form:

?n ↑C

where n is one of the error codes in Table 3-9 and ↑C indicates that control will pass to the Keyboard Monitor when a character is typed.

Table 3-9 EDITOR Error Codes

Error Code	Meaning
0	The EDITOR failed in reading from a device. An error occurred in the device handler; most likely a hardware malfunction.
1	The EDITOR failed in writing onto a device; generally a hardware malfunction.
2	A file close error occurred. The output file could not be closed; either the cassette reached an end-of-tape condition, or a sentinel file needs to be written before any new output files can be created on the cassette.

A ? occurs any time the EDITOR encounters a syntax error. In addition, the following error message may be printed by the EDITOR:

Message	Meaning
UNIT HAS OPENED FILE	Two files cannot be open on the same device at the same time.

During the editing of a file, the output cassette specified in the command string may become full before the editing process is completed. If this is the case and further writing is attempted on that cassette, an error occurs. The output file is closed and the message:

FULL*OUTPUT FILE-

is printed. The user must now indicate a new output cassette and file which will contain the text that would not fit on the first cassette and any further editing the user wishes to do. Since the contents of the text buffer are retained through this procedure, no text will be lost if this error occurs.

NOTE

If no output file is specified when this condition occurs, the EDITOR again requests an output file; this continues until the output designation is correctly specified.

Assuming the new output device is valid, the EDITOR will continue the operation which filled the old file, putting all output into the new output file. After editing is completed, the output files should be combined using the EDITOR. The entire process may then appear as follows:

```
.R EDIT
*INPUT FILE-0:IN
*OUTPUT FILE-1:OUT
#Y
#J
$STRING'
FULL *OUTPUT FILE-2:OUTEMP
#.L                                  Device 1: is full.
                                     2: is specified as the
        TAD STRING                   new output device
#.D                                  and editing continues.
#E
.
```

At this point the output "file" is 2 files--1:OUT, 2:OUTEMP. When output is split like this, the split may have occurred in the middle of a line. Therefore, the output files should never be edited separately as the split lines will then be lost. In a case such as this, the files should be combined with the EDITOR as follows:

```
.R EDIT/M
*INPUT FILE-1:OUT
*OUTPUT FILE-3:OUT
#E
*INPUT FILE-2:OUTEMP
*INPUT FILE-
```

The new file, OUT, may then be edited.

3.10 EDITOR DEMONSTRATION RUN

The following example illustrates both the use of the EDITOR to create a new file and a few of the commands available for editing. Sections of the printout are coded by letters which correspond with the explanations following the example.

```
A       .R EDIT
        *INPUT FILE-
        *OUTPUT FILE-0:PROG.PAL

        #A

        CHRPUT,0                /ACCEPTS CHAR IN AC AND
                SNA             /PACKS IT INTO OUTPUT BUFFER
B               JMP I CHRPUT    /IGNORES NULL
                CDF 0
                DCA SHELF
C               TAD WI\HAT1
                SPC
D               JMP PUT#1
                SNA CLO
                JMP PUT#2
                CMA
                DCA WHAT1

E       #.-3S
                SNA CLO\A

        #.L
                SNA CLA

F       #P

        #K

        #A
                TAD SHELF
                AND (360
                CLL RTL
G               .
                .
                .
                .
                .

H       #E
        .

I       .R EDIT
        *INPUT FILE-0:PROG.PAL
        *OUTPUT FILE-1:PROG.PAL

J       #R
        #$SPC'L
                SPC

K       #.S
                SPC\A

L       #E
        .
```

A	The user calls the EDITOR; the output file will be called PROG.PAL and will be stored on the default device--cassette drive 0. There is no input file since one will be created from the console terminal keyboard. The Append command is used to insert text into the empty buffer.
B	Text is inserted.
C	The user makes a mistake and uses the RUBOUT key to correct it
D	More text is added.
E	The user notices a typing mistake he has made several lines back in the text. He types a CTRL/FORM to finish the Append command, searchs for the illegal character, corrects it, and then lists the line.
F	The P command writes the current buffer into the output file placing a form feed after the last line. The K command deletes all text in the current buffer in preparation for a new page of text.
G	The user inserts new text using the Append command. When he is finished he types a CTRL/FORM to end the command.
H	The user closes the file; control returns to the Cassette Keyboard Monitor.
I	In looking over the listing, the user notices another mistake; he opens the file, calling it by the same name in both the input and output specification lines.
J	The Intra-Buffer Character String Search is used to locate the illegal instruction and list it.
K	The Single Character Search is used to find the letter to be corrected, and the RUBOUT key deletes it.
L	The file is closed and control again returns to the Keyboard Monitor.

CHAPTER 4
SYSTEM COPY

4.1 INTRODUCTION

The CAPS-8 System Copy (SYSCOP) program allows the user to copy individual files or all files from one cassette to another, giving him the ability to make multiple copies of a cassette, add files to a cassette, and "clean up" full cassettes so that they may become available for future use. System Copy transfers all non-empty files on the specified input cassette to the specified output cassette; space taken up by previously deleted files (*EMPTY files) is regained. (Single file transfers of ASCII files can be performed using the CAPS-8 EDITOR; see Chapter 3.)

4.2 CALLING AND USING SYSTEM COPY

To call SYSCOP from the System Cassette, the user types:

.R SYSCOP/Options

in response to the dot (.) printed by the Keyboard Monitor.

4.2.1 System Copy Options

There are three options available for use with System Copy; these options are discussed in Table 4-1. (Option usage is explained in Chapter 2, Section 2.4.3.)

Table 4-1 System Copy Options

Option	Meaning
/F	This option allows the user to transfer individual cassette files from one cassette to another. To use the /F option, the user responds to the request for input specification with the cassette drive number and the name of the file to be copied. If the user makes a typing error while entering the input specification, he can type CTRL/U to redo the entry.
/U	If the /U option is specified, drive 1 is zeroed and then drive 0 is copied to drive 1. (The /U option is especially useful for making copies of the System Cassette.) When the /U option is used, no further I/O specifications are necessary.
/Z	This option causes the output drive (indicated in the output specification line) to be zeroed before any copying begins.

4.2.2 Input and Output Specifications

Before indicating the input and output drives to be used for the copy operation, the user must ensure that the proper cassettes are mounted. The input cassette (the one to be copied) should be write-locked to protect the data. The output cassette (the one that will be the new copy) should be write-enabled to receive the data. When the input and output cassettes are mounted on the correct drives, the user is ready to begin the copy operation.

After SYSCOP has been called from the System Cassette, it asks for the input specification as follows:

```
IN-
```

The user responds with a single digit (0 through 7) specifying the input cassette drive number. A carriage return is not necessary. If the /F option was used, the user responds to the IN- query with the drive number and the name of the file to be copied; in this case, the user must also type a carriage return. In the following example, a file named ECHO is to be copied from drive 1.

```
IN-:ECHO
```

After the input specification has been entered, System Copy requests the output specification as follows:

```
OUT-
```

The user responds with a single digit (0 through 7) specifying the output drive number. The output drive number cannot be the same as the input drive number. If the user wishes to change the input/output specifications at this point, he may type a carriage return instead of the drive number after OUT- to return to the IN- message.

After both input and output drives have been indicated, the copy operation starts. All non-empty files on the input cassette are copied, in order, onto the output cassette. (If a file is to be copied onto a cassette under the same filename and extension as one already present on the cassette, it will still be copied; however, future reference to the file will cause the first file under that name to be accessed. To circumvent this condition, the user should first delete any old files or zero the output cassette.) When all files have been copied, control returns to the Keyboard Monitor.

Only two responses other than the digits 0 through 7 are accepted in reply to the input/output specification messages: carriage return and CTRL/C. Carriage return returns the user to the input specification message; CTRL/C returns the user to the Keyboard Monitor. Any other response is considered illegal. Illegal responses are neither accepted nor echoed by System Copy; System Copy simply waits for the user to type a legal response.

4.2.3 System Copy Example

In this example, the user wishes to make a copy of the System Cassette which is mounted on drive 0. One purpose of the copy operation is to regain wasted space being taken up by previously deleted files. A directory listing shows that the System Cassette currently contains the following files :

```
C2BOOT.BIN  Ø1/22/73
MONTOR.BIN  Ø1/22/73
SYSCOP.BIN  Ø1/25/73
*EMPTY.
EDIT  .BIN  Ø1/Ø2/73
PALC  .BIN  Ø1/Ø2/73
BASIC .BIN  Ø1/Ø2/73
*EMPTY.
*EMPTY.
*EMPTY.
*EMPTY.
ABC   .     Ø1/22/73
```

The user mounts a write-enabled cassette on drive 2 and rewinds the tape. He than calls System Copy as follows :

```
.R SYSCOP/Z
```

The /Z option will zero the cassette mounted on the cassette drive specified in the OUT- message (drive 2), leaving only the sentinel file on the cassette. System Copy then requests the input and output drive numbers and the user responds as follows:

```
IN-Ø
OUT-2
```

The copy operation starts. If System Copy detects any problems during the copy operation, it prints one of the error messages explained in Section 4.3. A successful copy operation returns control to the Keyboard Monitor. The user can then issue a DIrectory command to ensure that all files were copied correctly. In this example, a successful copy operation should produce the following directory listing:

```
C2BOOT.BIN  Ø1/22/73
MONTOR.BIN  Ø1/22/73
SYSCOP.BIN  Ø1/25/73
EDIT  .BIN  Ø1/Ø2/73
PALC  .BIN  Ø1/Ø2/73
BASIC .BIN  Ø1/Ø2/73
ABC   .     Ø1/22/73
```

4.3 SYSTEM COPY ERROR MESSAGES

Errors which occur during a System Copy operation may be of two types: user errors and cassette errors. User errors may be corrected with the appropriate action as detailed in Table 4-2. Cassette errors normally require the user to use another cassette (for either input or output) to complete the copy operation. Control does not return to the Keyboard Monitor when a System Copy error occurs. The user may use CTRL/C to return to the Monitor if he cannot correct the indicated error.

Table 4-2 System Copy Error Messages

Message	Meaning
INPUT ERROR ON UNIT n	An input error has occurred on the cassette drive specified. The user should try the copy operation using another cassette.
UNIT n NOT READY	There is either no cassette on the cassette drive specified or no such drive exists.
UNIT n WRITE LOCKED	The user tried to write data when the write protect tab of the cassette on the drive specified was write-locked.
OUTPUT ERROR ON UNIT n	An output error has occurred on the cassette drive specified. The user should try the copy operation using another cassette.

CHAPTER 5
PALC

5.1 INTRODUCTION

PALC (an acronym for Program Assembly Language for Cassette) is an 8K 2-pass assembler (with an optional third pass) designed for the CAPS-8 System. A program written in PALC source language is translated by the assembler into a binary file in two passes. Pass 1 reads the input file and sets up the symbol table; pass 2 reads the input file and uses the symbol table created in pass 1 to generate the binary (object) file. The binary file may then be loaded into memory using the Cassette Keyboard Monitor.

PALC allows I/O using any CAPS-8 device which handles ASCII text. It is called from the System Cassette using the Keyboard Monitor Run command, accepts input generated by the CAPS-8 EDITOR, and will generate output acceptable for use with both the Monitor Load and Run commands.

5.2 CALLING AND USING PALC

The user calls PALC from the System Cassette by typing:

```
.R PALC/Options
```

PALC responds by printing:

```
-INPUT FILES
*
```

The user enters his input cassette drive number and filename in answer to the asterisk printed by PALC; a total of three input specifications are allowed, so that the input interaction may appear as follows:

```
-INPUT FILES
*1:TRA.PAL
*0:TRB.PAL
*1:TRC.PAL
```

Usually input files will contain the extension .PAL (see Chapter 2, Section 2.2.2), and PALC will assume this extension unless the user explicitly designates another. Thus in the above example the user may have responded by typing only 1:TRA, 0:TRB, and 1:TRC, in which case PALC would automatically assume and echo the .PAL extension.

If the filename contains an extension other than .PAL, the user must specify this extension when entering the input. For example:

```
-INPUT FILES
*1:FAIL.1
*2:TABLE.ASC
*0:SHOR.
```

In the case of the third input file (SHOR.) an extension is not to be indicated. If the user wants to prevent PALC from assuming .PAL, he must be sure to include a period in the input line; otherwise PALC will append .PAL and look for the filename with that extension.

If the user does not specify a drive number in his input line, the default device--drive 0--is assumed. PALC will automatically insert 0: in the input line before echoing the filename as the user has entered it. For example:

```
*0:FLOA.PAL
```

The user actually typed only the characters FLOA; PALC assumed both the drive number 0 and the .PAL extension and correctly inserted these in the I/O line before echoing the complete line.

A carriage return typed in response to any of the asterisks indicates that there are no more input files.

After the input specifications have been entered, PALC requests the binary output as follows:

```
-BINARY FILE
*
```

The user responds similarly here by indicating an output drive number and filename. Only one binary file is allowed and it should have the extension .BIN (since the Monitor Run and Load commands assume this extension). If the user wants his binary file to be called by the same name as the first input file he need only type the drive number, a colon, and a carriage return. PALC will echo this, adding the filename with a .BIN extension. For example:

```
-INPUT FILES
*1:OPEN.PAL
*
-BINARY FILE
*0:OPEN.BIN
```

As in the input line, drive 0 and the extension (.BIN) are assumed if the user fails to specify them, and a response of only a carriage return indicates that no binary file is to be produced.

Once the binary output line has been answered, PALC prints:

```
-LIST TO
*
```

The user has a choice of sending his output listing to either the console terminal or the line printer. To send output to the console terminal the user types the characters TTY in response to the asterisk as follows:

```
-LIST TO
*TTY
```

To send output to the line printer, the user responds by typing LPT:

```
-LIST TO
*LPT
```

A response of a carriage return indicates that no listing is to be produced.

During a PALC assembly only one listing is produced and it may be sent to only one device, either the line printer or console terminal. A second listing must be produced by another assembly.

If more cassettes are to be involved in the assembly than the user has TU60 unit drives, a certain procedure must be followed during the assembly process. For example, assume the user has only one TU60 dual cassette unit, and 3 input files are stored on individual cassettes. His I/O specification is as follows:

```
-INPUT FILES
*1:F1.PAL
*0:F2.PAL
*0:F3.PAL
-BINARY FILE
*1:RESLT.BIN
-LIST TO
*LPT
```

PALC is a 3-pass assembler, therefore all three input files will be referenced 3 times. Assume the user has mounted 1:F1.PAL on drive 1, and 0:F2.PAL on drive 0; assembly begins. First the file F1 is processed, then F2. After assembly of F2 PALC looks for F3, but since the file is on a third cassette which is not mounted, the assembly pauses and PALC prints:

```
MOUNT F3.PAL?
```

This pause in the assembly allows the user to dismount a cassette and replace it with the cassette containing the file F3.PAL. The user then responds to the above I/O line with the drive number on which he has mounted the new cassette (assume 0), as follows:

```
MOUNT F3.PAL?0
```

If the response is valid, PALC responds by typing a CR/LF and continues pass 1 of the assembly. (An invalid response causes PALC to print a ?; the user may then type the correct response.)

When pass 1 is completed PALC automatically begins the second pass, which creates the binary file. The binary output file specification must now be made. Regardless of the output specification indicated in the initial dialogue, PALC pauses and asks:

```
MOUNT RESLT.BIN?
```

The user must mount the output cassette which is to contain the binary file and respond with the drive number on which he has mounted it. Assume he decides to mount the cassette on drive 0. He replaces the cassette currently on that drive (containing F3.PAL) with the new cassette and responds to the command line as follows:

```
MOUNT RESLT.BIN?0
```

PALC then opens the binary file on this cassette and prints:

```
BINARY FILE OPENED ON 0
```

NOTE

> The cassette used for binary output may not contain any of the input files. Under no circumstances should the cassette containing the binary file be removed from the drive until pass 2 is completely finished. PALC will indicate completion of the pass by printing the message, "BINARY FILE CLOSED".

After specification of the binary output file, PALC continues pass 2 of the assembly by processing the first input file, F1.PAL, currently on drive 1. After this file is processed, PALC pauses and asks:

```
MOUNT F2.PAL?
```

Since the binary file being created on drive 0 is only partially complete at this point, the user must not remove the cassette from that drive. He must instead remove the cassette from drive 1 and replace it with the cassette containing F2.PAL. He then types 1 in response to the I/O line and assembly continues until F3.PAL is needed. PALC again pauses and asks:

```
MOUNT F3.PAL?
```

Again the user replaces the cassette on drive 1 with the appropriate one, correctly answers the I/O line, and assembly continues.

Once pass 2 is done, pass 3--the listing pass--must be processed. Drive 0 may again be used for input, and assembly of input files continues in the same manner as during passes 1 and 2.

The procedure of mounting and dismounting cassettes may be repeated as many times as necessary until all input files are processed and the desired output produced. If an I/O error occurs during any of the three passes or if an output cassette becomes full, the user must restart the assembly beginning with pass 1.

NOTE

When the assembly is complete, PALC prints ↑C (CTRL/C). The user then mounts the system cassette and types CTRL/C to return to the Monitor.

5.2.1 PALC Options

Table 5-1 lists the options available in PALC which may be indicated in the Monitor Run or Load specification line.

Table 5-1 PALC Options

Option	Meaning
/D	Generate a DDT-compatible symbol table (applicable only if a listing file is specified).
/H	Generate non-paginated output. Header, page numbers and page format are suppressed (applicable only if a listing file is specified).
/K	Used in assembling very large programs; causes system containing 12K or more of memory to use fields 2 and up as symbol table storage.
/N	Generate the symbol table, but not the listing (applicable only if a listing file is specified; the /H option is assumed).
/S	Omit the symbol table normally generated with the listing (applicable only if a listing file is specified).
/T	Output a carriage return/line feed in place of the form feed character(s) in the program (applicable only if a listing file is specified).

5.3 CHARACTER SET

The following characters are acceptable as input to PALC:

1. The alphabetic characters: A through Z
2. The numeric characters: 0 through 9
3. The characters described in following sections as special characters and operators

4. Characters which are ignored during assembly such as LINE FEED, FORM FEED, and RUBOUT

All other characters are illegal (except when used in a comment) and cause the error message:

IC AT nnnn

to be printed during pass 1; nnnn represents the location at which the illegal character occurred. (As assembly proceeds, each instruction is assigned a location determined by the current location counter (see Section 5.7.3). When an illegal character or any other error is encountered during assembly, the value of the current location counter is returned in the error message.) Illegal characters do not generally cause assembly to halt. If an illegal character occurs in the middle of a symbol, the symbol is terminated at that point.

5.4 STATEMENTS

PALC source programs are usually prepared on the console terminal (using the CAPS-8 EDITOR) as a sequence of statements. Each statement is written on a single line and is terminated by typing the RETURN key.

There are four types of elements in a PALC statement which are identified by the order of their appearance in the statement and by the separating (or delimiting) character which follows or precedes the element. These are:

label, instruction operand /comment

A statement must contain at least one of these elements and may contain all four types. The assembler interprets and processes the statements, generating one or more binary instructions or data words, or performing an assembly process.

5.4.1 Labels

A label is the symbolic name created by the programmer to identify the location of a statement in the program. If present, the label is written first in a statement. It must begin with an alphabetic character, contain only alphanumeric characters, and be terminated by a comma; there must be no intervening spaces between any of the characters and the comma.

5.4.2 Instructions

An instruction may be one or more of the mnemonic machine instructions or a pseudo-operation which directs assembly processing. (Assembly pseudo-ops are described later in this chapter; Appendix C summarizes both the mnemonic machine instructions and pseudo-ops used by PALC.) Instructions are terminated with one or more spaces (or tabs if an

operand follows) or with a semicolon, slash, or carriage return, as described in Section 5.5.3.

5.4.3 Operands

Operands are the octal or symbolic addresses of an assembly language instruction or the argument of a pseudo-operator, and can be any expression. In each case, interpretation of an operand depends upon the instruction or the pseudo-op. Operands are terminated by a semicolon, slash, or carriage return.

5.4.4 Comments

The programmer may add notes or comments to a statement by separating these from the remainder of the line with a slash. Such comments do not affect assembly processing or program execution but are useful in the program listing for later analysis or debugging. The assembler ignores everything from the slash to the next carriage return. (For an example see Section 5.5.3, Statement Terminators.)

It is possible to have only a carriage return on a line, resulting in a blank line in the final listing. No error message is given.

5.5 FORMAT EFFECTORS

The following characters are useful in controlling the format of an assembly listing. They allow a neat readable listing to be produced by providing a means of spacing through the program.

5.5.1 Form Feed

The form feed code causes the assembler to output blank lines in order to skip to a new page in the output listing during pass 3; this is useful in creating a page-by-page listing. The form feed is generated by typing a CTRL/L on the console terminal.

5.5.2 Tabulations

Tabulations are used in the body of a source program to separate fields into columns (for details refer to Chapter 3). For example, a line written:

```
GO,TAD TOTAL/MAIN LOOP
```

is much easier to read if tabs are inserted to form:

```
GO,     TAD TOTAL          /MAIN LOOP
```

5.5.3 Statement Terminators

The RETURN key is used to terminate a statement and causes a line feed/carriage return combination to occur in the listing. The semicolon (;) may also be used as a statement terminator and is considered identical to a carriage return except that it will not terminate a comment. For example:

```
TAD A   /THIS IS A COMMENT; TAD B
```

The entire expression between the slash (/) and the carriage return is considered a comment. Thus in this case the assembler ignores the TAD B.

If, for example, the user wishes to write a sequence of instructions to rotate the contents of the accumulator and link six places to the right, it might look like the following:

```
RTR
RTR
RTR
```

However, the programmer can alternatively place all three instructions on a single line by separating them with the special character semicolon (;) and terminating the entire line with a carriage return. The above sequence of instructions can then be written:

```
RTR;RTR;RTR
```

These multi-statement lines are particularly useful when setting aside a section of data storage for use during processing. For example, a 4-word cleared block could be reserved by specifying either of the following:

```
LIST,   0;      0;      0;      0
```

or

```
LIST, 0
      0
      0
      0
```

Either format may be used to input data words (data words may be in the form of numbers, symbols, or expressions, explained next.) Each of the following lines generates one storage word in the object program:

```
DATA, 7777
      A+C-B
      S
      123+B2
```

5.6 NUMBERS

Any sequence of digits delimited by either a SPACE, TAB, semicolon, or carriage return forms a number. PALC initally interprets numbers in octal (base 8). This base can be changed to decimal using a special pseudo-operator (discussed in Section 5.10.2). Numbers are used in conjunction with symbols to form expressions.

5.7 SYMBOLS

A symbol is a string of alphanumeric characters beginning with a letter and delimited by a non-alphanumeric character. Although a symbol may be any length only the first six characters are recognized; since additional characters are ignored, symbols which are identical in their first six characters are considered identical.

5.7.1 Permanent Symbols

The assembler contains a table (called its permanent symbol table) which lists the symbols for all PDP-8 pseudo-op codes, memory reference instructions, operate and IOT (Input/Output Transfer) instructions. These instructions are symbols which are permanently defined by PALC and need no further definition by the user; they are summarized in Appendix C. For example:

HLT	This is a symbolic instruction assigned the value 7402 by the assembler and stored in its permanent symbol table.

5.7.2 User-Defined Symbols

All symbols not defined by the assembler (and represented in its permanent symbol table) must be defined within the source program.

A symbol may be used as a statement label, in which case it is assigned a value equal to the current location counter; it is called a symbolic address and can be used as an operand or as a reference to an instruction. Permanent symbols (instructions, special characters, and pseudo-ops) may not be used as symbolic addresses.

The following are examples of legal symbolic addresses:

```
ADDR,
TOTAL,
SUM,
A1,
```

The following are illegal symbolic addresses:

```
AD>M,       (contains an illegal character)
7ABC,       (first character must be alphabetic)
LA BEL,     (must not contain imbedded spaces)
D+TAG,      (contains a legal but non-alphanumeric character)
LABEL ,     (must be terminated by a comma with no
                intervening spaces)
```

5.7.3 Current Location Counter

As source statements are processed, PALC assigns consecutive memory addresses to the instructions and data words of the object program. The current location counter contains the address in which the next word of object code will be assembled and is automatically incremented each time a memory location is assigned. A statement which generates a single object program storage word increments the location counter by one. Another statement might generate six storage words, incrementing the location counter by six.

The user sets or resets the location counter by typing an asterisk followed by the octal absolute address value in which the next program word is to be stored. If the origin is not set by the user, PALC begins assigning addresses at location 200.

```
          *300                    /SET LOCATION COUNTER TO 300
TAG,      CLA
          JMP A
B,        Ø
A,        DCA B
          .
          .
          .
```

The symbol TAG (in the preceding example) is assigned a value of 0300, the symbol B a value of 0302, and the symbol A a value of 0303. If a symbol is defined more than once in this manner, the assembler will print the illegal definition diagnostic:

ID address

where address is the value of the location counter at the second occurrence of the symbol definition. The symbol is not redefined. (For an explanation of diagnostic messages refer to Section 5.14 PALC Error Conditions.) For example:

```
          *300
START,    TAD A
          DCA COUNTER
CONTIN,   JMS LEAVE
          JMP START
A,        -74
COUNTER,  Ø
START,    CLA CLL
          .
          .
          .
```

The symbol START would have a value of 0300, the symbol CONTIN would have a value of 0302, the symbol A would have a value of 0304, the symbol COUNTER (considered COUNTE by the assembler) would have a value of 0305. When the assembler processed the next line it would print (during pass 1):

```
IR COUNTE+0001
```

Since the first pass of PALC is used to define all symbols, the assembler will print a diagnostic during pass 2 if reference is made to an undefined symbol. For example:

```
                *7170
A,              TAD C
                CLA CMA
                HLT
                JMP A1          The dollar sign must terminate
C,              0               all PDP-8 assembly programs.
                $
```

This would produce the undefined symbol diagnostic:

```
US A+0003
```

5.7.4 Symbol Table

Initially, the assembler's symbol table contains the mnemonic op-codes of the machine instructions and the assembler pseudo-op codes as listed in Appendix C; this is its permanent symbol table. As the source program is processed, user-defined symbols along with their binary values are added to the symbol table. The symbol table is listed in alphabetical order at the end of pass 3.

During pass 1, if PALC detects that the symbol table is full (in other words, there is no more memory space in which to store symbols and their associated values), the symbol table exceeded diagnostic is printed:

SE address

The assembler then prints ↑C and waits for a response from the user. By typing ↑C the user can return control to the Monitor. If the system contains more than 8K of memory, the user may choose the /K option with the Run command (see Section 5.2.1), or more address arithmetic may be used to reduce the number of symbols. It is also possible to segment a program and assemble the segments separately, taking care to generate proper links between the segments. (See Section 5.11.) PALC's symbol capacity is 768 symbols. The permanent symbol table contains 69 symbols, leaving space for 699 possible user-defined symbols. Each additional 4K allows 768 new symbols.

Section 5.10.12 provides instructions concerning altering PALC's permanent symbol table should the user wish to add instructions more suited to his programming needs.

5.7.5 Direct Assignment Statements

The programmer may insert new symbols with their assigned values directly into the symbol table by using a direct assignment statement of the form:

SYMBOL=VALUE

VALUE may be a number or expression. No spaces or tabs may appear between the symbol to the left of the equal sign and the equal sign itself. The following are examples of direct assignment statements:

```
A=6
EXIT=JMP I Ø
C=A+B
```

All symbols to the right of the equal sign must be already defined. The symbol to the left of the equal sign is subject to the same restrictions as a symbolic address, and its associated value is stored in the user's symbol table. The use of the equal sign does not increment the location counter; it is, rather, an instruction to the assembler itself.

A direct assignment statement may also equate a new symbol to the value assigned to a previously defined symbol. In this case, the two symbols share the same memory location.

```
BETA=17
GAMMA=BETA
```

The new symbol, GAMMA, is entered into the user's symbol table with the value 17.

The value assigned to a symbol may be changed as follows:

```
ALPHA=5
ALPHA=7
```

The second line of code shown changes the value assigned to ALPHA from 5 to 7. (This is legal but will generate an RD error message, explained below.)

Symbols defined by use of the equal sign may be used in any valid expression. For example:

```
        *200
A=100           /DOES NOT UPDATE CLC
B=400           /DOES NOT UPDATE CLC
A+B             /THE VALUE 500 IS ASSEMBLED AT LOC. 200
TAD A           /THE VALUE 1200 IS ASSEMBLED AT LOC. 201
```

If the symbol to the left of the equal sign has already been defined, the redefinition diagnostic:

RD address

will be printed as a warning, where address is the value of the location counter at the point of redefinition. The new value will be

stored in the symbol table; for example:

```
CLA=7600
```

will cause the diagnostic:

```
RD +0200
```

Whenever CLA is used after this point, it will have the value 7600.

5.7.6 Symbolic Instructions

Symbols used as instructions must be predefined by the assembler or the programmer. If a statement has no label, the instructions may appear first in the statement and must be terminated by a space, tab, semicolon, slash, or carriage return. The following are examples of legal instructions:

```
TAD       (a mnemonic machine instruction)
PAGE      (an assembler pseudo-op)
ZIP       (an instruction defined by the user)
```

5.7.7 Symbolic Operands

Symbols used as operands normally have a value defined by the user. The assembler allows symbolic references to instructions or data defined elsewhere in the program. Operands may be numbers or expressions. For example:

```
TOTAL, TAD AC1 + TAG
```

The values of the two symbols AC1 and TAG (already defined by the user) are combined by a 2's complement add (see Section 5.8.1, Operators). This value is then used as the address of the operand.

5.7.8 Internal Symbol Representation For PALC

Each permanent and user-defined symbol occupies four words in the symbol table storage area. A PDP-8 instruction has an operation code of three bits as well as an indirect bit, a page bit, and seven address bits. The PALC assembler distinguishes between pseudo-ops, memory reference instructions, other permanent symbols, and user-defined symbols in the symbol table.

5.8 EXPRESSIONS

Expressions are formed by the combination of symbols, numbers, and certain characters called operators, which cause specific arithmetic operations to be performed. An expression is terminated by either a comma, carriage return, or semicolon.

5.8.1 Operators

There are seven characters in PALC which act as operators:

+	Two's complement addition
-	Two's complement subtraction
↑	Multiplication (unsigned, 12-bit)
%	Division (unsigned, 12-bit)
!	Boolean inclusive OR
&	Boolean AND
SPACE	Treated as a Boolean inclusive OR except in a memory reference instruction

Two's complement addition and subtraction are explained in detail in Chapter 1 of INTRODUCTION TO PROGRAMMING; the user should refer to that handbook if he wishes more information. No checks for overflow are made during assembly, and any overflow bits are lost from the high order end. For example:

7755+24 will give a result of 1

The operators + and - may be used freely as prefix operators.

Multiplication is accomplished by repeated addition. No checks for sign or overflow are made. All 12 bits of each factor are considered as magnitude. For example:

3000↑2 will give a result of 6000

Division is accomplished by repeated subtraction. The number of subtractions which are performed is the quotient. The remainder is not saved and no checks are made for sign. Division by 0 will arbitrarily yield a result of 0. For example:

7000%1000 will yield a result of 7

This could be written as:

-1000%1000

in this case the answer might be expected to be -1 (7777), but all 12 bits are considered as magnitude and the result is still 7.

Use of the multiplication and division operators requires an attention to sign on the part of the programmer beyond that which is required for simple addition and subtraction. The following table of examples is given for reference.

Table 5-2 Use of Operators

Expression	Also written as:	Result
7777+2	-1+2	+1
7776-3	-2-3	7773 or -5
0↑2		0
2↑0		0
1000↑7		7000 or -1000
0%12		0
12%0		0
7777%1	-1%1	7777 or -1
7000%1000	-1000%1000	7
1%2		0

The ! operator causes a Boolean inclusive OR to be performed bit by bit between the left-hand term and the right-hand term. (The inclusive OR is explained in Chapter 1 of INTRODUCTION TO PROGRAMMING.) For example:

```
if A=1 and B=2
then A!B=0003
```

The & operator causes a Boolean AND to be performed bit by bit between the left and right values. The operation is the same as that indicated by the memory reference instruction AND.

SPACE has special significance depending on the context in which it is used. When it is used to separate two permanent symbols or two user-defined symbols, as in the following example:

```
    SMA CLA
```

it causes an inclusive OR to be performed between them. In this case, SMA=7500 and CLA=7600. The expression SMA CLA is assembled as 7700. When SPACE is used following pseudo-operators and memory reference instructions, it merely delimits the symbol.

User-defined symbols are treated as operate instructions. For example:

```
      A=333
      *222
B,    CLA
```

Possible expressions and their values using the symbols just defined are shown below. Notice that the assembler reduces each expression to one 4-digit (octal) word:

```
A       0333
B       0222
A+B     0555
A-B     0111
-A      7445
1-B     7557
B-1     0221
A!B     0333    (an inclusive OR is performed)
-71     7707
```

If the information generated is to be loaded, the current location counter is incremented. For example:

```
B-7;A+4;A-B
```

produces three words of information; the current location counter is incremented after each expression. The statement:

```
HALT=HLT CLA
```

produces no information to be loaded (it produces an association in the symbol table) and hence does not increment the current location counter.

```
        *4721
TEMP,
TEM2,   Ø
```

The location counter is not incremented after the line TEMP,; the two symbols TEMP and TEM2 are assigned the same value, in this case 4721.

Since a PDP-8 instruction has an operation code of three bits as well as an indirect bit, a page bit, and seven address bits, the assembler must combine memory reference instructions in a manner somewhat differently from the way in which it combines operate or IOT instructions. The assembler differentiates between the symbols in its permanent symbol table and user-defined symbols. The following symbols are used as memory reference instructions:

AND	0000	Logical AND
TAD	1000	Two's complement addition
ISZ	2000	Increment and skip if zero
DCA	3000	Deposit and clear accumulator
JMS	4000	Jump to subroutine
JMP	5000	Jump

When the assembler has processed one of these symbols, the space following it acts as an address field delimiter.

```
        *41ØØ
        JMP A
A,      CLA
```

A has the value 4101, JMP has the value 5000, and the space acts as a field delimiter. These symbols are represented as follows:

```
A       100 001 000 001
JMP     101 000 000 000
```

The seven address bits of A are taken, e.g.:

```
000 001 000 001
```

The remaining bits of the address are tested to see if they are zeros (page zero reference); if they are not, the current page bit is set:

000 011 000 001

The operation code is then ORed into the JMP expression to form;

101 011 000 001

or, written more concisely in octal:

5301

In addition to the above tests, the page bits of the address field are compared with the page bits of the current location counter. If the page bits of the address field are nonzero and do not equal the page bits of the current location counter, an out-of-page reference is being attempted and the assembler will take action as described in Section 5.11, Link Generation and Storage.

5.8.2 Special Characters

In addition to the operators described in the previous section, PALC recognizes several special characters which serve specific functions in the assembly process. These characters are:

```
=   equal sign
,   comma
*   asterisk
.   dot
"   double quote
( ) parentheses
[]  square brackets
/   slash
;   semicolon
<>  angle brackets
$   dollar sign
```

The equal sign, comma, asterisk, slash, and semicolon have been previously described. The remainder will be described next.

The special character dot (.) always has a value equal to the value of the current location counter. It may be used as any integer or symbol (except to the left of an equal sign), and must be preceded by a space when used as an operand. For example:

```
*200
JMP  .+2
```

is equivalent to JMP 0202. Also,

```
*300
.+2400
```

will produce in location 0300 the quantity 2700. Consider:

```
*2200
CALL=JMS I .
0027
```

The second line (CALL=JMS I.) does not increment the current location counter; therefore, 0027 is placed in location 2200 and CALL is placed in the user's symbol table with an associated value of 4600 (the octal equivalent of JMS I).

If a single character is preceded by a double quote ("), the 8-bit value of ASCII code for the character is used rather than interpreting the character as a symbol (ASCII codes are listed in Appendix A). For example:

```
CLA
TAD      ("A
```

The constant 0301 is placed in the accumulator.

The code:

```
".
```

will be assembled as 0256. The character must not be a carriage return or one of the characters which is ignored on input (discussed at the end of this section).

Left and right parentheses () enclose a current page literal (closing member is optional).

```
*200
          .
          .
          .
          CLA
          TAD INDEX
          TAD (2)
          DCA INDEX
          .
          .
          .
```

The left parenthesis is a signal to the assembler that the expression following is to be evaluated and assigned a word in the constants table of the current page. This is the same table in which the indirect address linkages are stored. In the above example, the quantity 2 is stored in a word in the linkage and literals list beginning at the top of the current memory page. The instruction in which the literal appears is encoded with an address referring to the address of the literal. A literal is assigned to storage the first time it is encountered; subsequent reference to that literal from the current page is made to the same register. The use of literals frees symbol storage space for variables and makes programs much more readable.

If the programmer wishes to assign literals to page zero rather than to the current page, he may use square brackets, [and] , in place of parentheses. This enables the programmer to reference a single literal from any page of memory. For example:

```
*200
          TAD [2]
          .
          .
          .
*500
          TAD [2]
          .
          .
          .
```

The closing member is optional. Literals may take the following forms: constant term, variable term, instruction, expression, or another literal.

NOTE

Literals can be nested, for example:

```
*200Ø
        TAD (TAD (3Ø
```

This type of nesting may be continued in some cases to as many as 6 levels, depending on the number of other literals on the page and the complexity of the expressions within the nest. If the limits of the assembler are reached, the error messages BE (too many levels of nesting) or PE (too many literals) will result.

Angle brackets are used as conditional delimiters. The code enclosed in the angle brackets is to be assembled or ignored contingent upon the definition of the symbol or value of the expression within the angle brackets. (The IFDEF, IFNDEF, IFZERO, and IFNZRO pseudo-operators are used with angle brackets and are described in Section 5.10.9.)

The dollar sign character ($) is mandatory at the end of a program and is interpreted as an unconditional end-of-pass. It may however occur in a text string, comment or " term, in which case it is interpreted in the same manner as any other character.

The following characters are handled by the assembler for the pass 3 listing, but are otherwise ignored:

FORM FEED	Used to skip to a new page
LINE FEED	Used to create a line spacing without causing a carriage return
RUBOUT	Used by the EDITOR to allow corrections in the input file.

Nonprinting characters include:

SPACE
TAB
RETURN

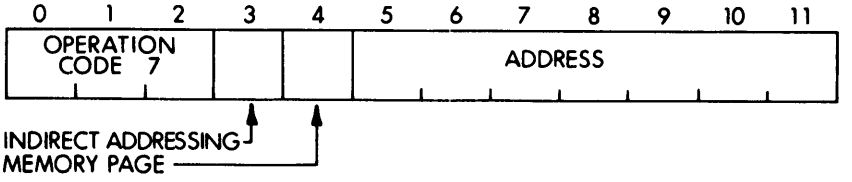
These characters are used for format control and have been previously explained in Section 5.5.

5.9 INSTRUCTIONS

There are two basic groups of instructions: memory reference and microinstructions. Memory reference instructions require an operand, microinstructions do not.

5.9.1 Memory Reference Instructions

In PDP-8 computers, some instructions require a reference to memory. They are appropriately designated memory reference instructions, and take the following format:



Memory Reference Bit Instructions

Bits 0 through 2 contain the operation code of the instruction to be performed. Bit 3 tells the computer if the instruction is indirect (see Section 5.9.2). Bit 4 tells the computer if the instruction is referencing the current page or page zero. This leaves bits 5 through 11 (7 bits) to specify an address. In these 7 bits, 200 octal (128 decimal) locations can be specified; the page bit increases accessible locations to 400 octal or 256 decimal. For a list of the memory reference instructions and their codes, see Appendix C.

In PALC a memory reference instruction must be followed by a space(s) or tab(s), an optional I or Z designation, and any valid expression. It may be defined with the FIXMRI instruction as explained in Section 5.10.12, Altering the Permanent Symbol Table. Permanent symbols may be defined using the FIXTAB instruction and may be used in address fields as shown below:

```
A=1234
FIXTAB
TAD A
```

5.9.2 Indirect Addressing

When the character I appears in a statement between a memory reference instruction and an operand, the operand is interpreted as the address (or location) containing the address of the operand to be used in the current statement. Consider:

```
TAD 40
```

which is a direct address statement, where 40 is interpreted as the location on page zero containing the quantity to be added to the accumulator. References to locations on the current page and page zero may be done directly. An alternate way to note the page zero reference is with the letter Z, as follows:

```
TAD Z 40
```

This is an optional notation, not differing in effect from the previous example. Thus, if location 40 contains 0432, then 0432 is added to the accumulator. Now consider:

```
TAD I 40
```

which is an indirect address statement, where 40 is interpreted as the address of the location containing the quantity to be added to the accumulator. Thus, if location 40 contains 0432, and location 432 contains 0456, then 456 is added to the accumulator.

NOTE

Because the letter I is used to indicate indirect addressing, it is never used as a variable. Likewise the letter Z, which is sometimes used to indicate a page zero reference, is never used as a variable.

5.9.3 Microinstructions

Microinstructions are divided into two groups: operate and Input/Output Transfer (IOT) microinstructions. Operate microinstructions are further subdivided into Group 1, Group 2, and Group 3 designations.

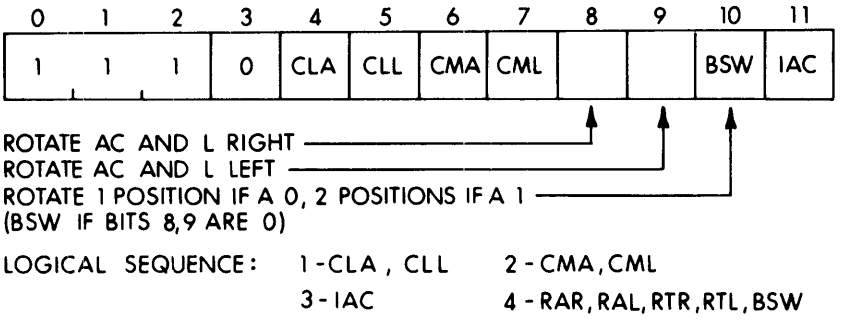
NOTE

If a programmer mistakenly specifies an illegal combination of micro-instructions, the assembler will perform an inclusive OR between them; for example:

```
    CLL SKP is interpreted as SPA
(7100) (7410)               (7510)
```

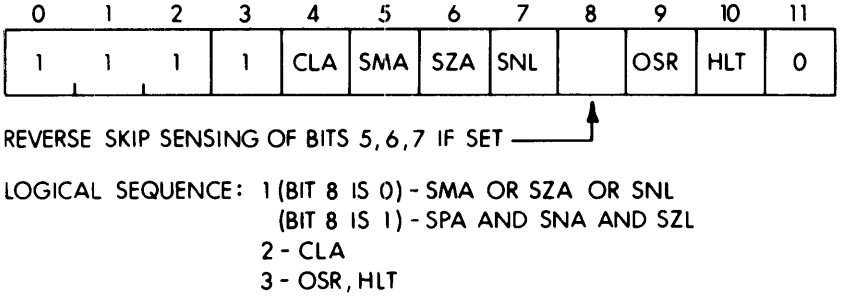
Operate Microinstructions

Within the operate group, there are three groups of microinstructions which cannot be mixed. Group 1 microinstructions perform clear, complement, rotate and increment operations, and are designated by the presence of a 0 in bit 3 of the machine instruction word.



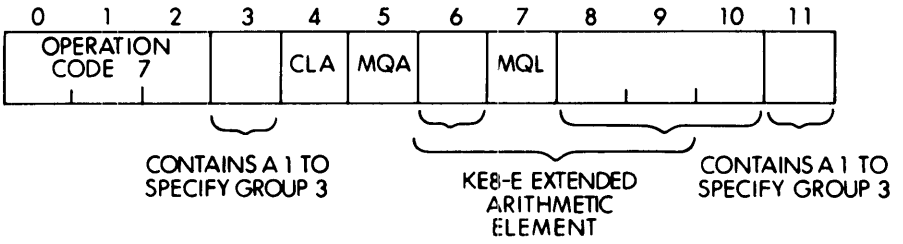
Group 1 Operate Microinstruction Bit Assignments

Group 2 microinstructions check the contents of the accumulator and link and, based on the check, continue to or skip the next instruction. Group 2 microinstructions are identified by the presence of a 1 in bit 3 and a 0 in bit 11 of the machine instruction word.



Group 2 Operate Microinstruction Bit Assignments

Group 3 instructions reference the MQ register. They are differentiated from Group 2 instructions by the presence of a 1 in bits 3 and 11. The other bits are part of a hardware arithmetic option.



Group 3 Operate Microinstruction Bit Assignments

Group 1 and Group 2 microinstructions cannot be combined since bit 3 determines either one or the other.

Within Group 2, there are two groups of skip instructions. They can be referred to as the OR group and the AND group.

OR Group	AND Group
SMA	SPA
SZA	SNA
SNL	SZL

The OR group is designated by a 0 in bit 8, and the AND group by a 1 in bit 8. OR and AND group instructions cannot be combined since bit 8 determines either one or the other.

If the programmer does combine legal skip instructions, it is important to note the conditions under which a skip may occur.

1. OR Group--If these skips are combined in a statement, the inclusive OR of the conditions determines the skip. For example:

   ```
   SZA SNL
   ```

   The next statement is skipped if the accumulator contains 0000 or the link is a 1 or both.

2. AND Group--If the skips are combined in a statement, the logical AND of the conditions determines the skip. For example:

   ```
   SNA SZL
   ```

   The next statement is skipped only if the accumulator differs from 0000 and the link is 0.

Input/Output Transfer Microinstructions

These microinstructions initiate operation of peripheral equipment and effect an information transfer between the central processor and the Input/Output device(s); i.e., cassettes, console terminal, and line printer. The Permanent Symbol Table in Appendix C lists PALC's IOT's.

5.9.4 Autoindexing

Interpage references are often necessary for obtaining operands when processing large amounts of data. The PDP-8 computers have facilities to ease the addressing of this data. When one of the absolute locations from 10 to 17 (octal) is indirectly addressed, the contents of the location is incremented before it is used as an address and the incremented number is left in the location. This allows the programmer to address consecutive memory locations using a minimum of statements.

It must be remembered that initially these locations (10 to 17 on page 0) must be set to one less than the first desired address. Because of their characteristics, these locations are called autoindex registers. No incrementation takes place when locations 10 to 17 are addressed directly. For example, if the instruction to be executed next is in location 300 and the data to be referenced is on the page starting at location 5000, autoindex register 10 can be used to address the data as follows:

```
0276    1377    TAD C4777       /=5000-1
0277    3010    DCA 10          /SET UP AUTO INDEX
0300    1410    TAD I 10        /INCREMENT TO 5000
.       .       .               /BEFORE USE AS AN ADDRESS
.       .       .
.       .       .
0377    4777    C4777,4777
```

When the instruction in location 300 is executed, the contents of location 10 will be incremented to 5000 and the contents of location 5000 will be added to the contents of the accumulator. When the instruction TAD I 10 is executed again, the contents of location 5001 will be added to the accumulator, and so on.

5.10 PSEUDO-OPERATORS

The programmer uses pseudo-operators to direct the assembler to perform certain tasks or to interpret subsequent coding in a certain manner. Some pseudo-ops generate storage words in the object program, other pseudo-ops direct the assembler how to proceed with the assembly. Pseudo-ops are maintained in the permanent symbol table.

The function of each PALC pseudo-op is described below.

5.10.1 Indirect and Page Zero Addressing

The pseudo-operators I and Z are used to specify the type of addressing to be performed. These have been previously discussed in Section 5.9.2.

5.10.2 Radix Control

Numbers used in a source program are initially considered to be octal numbers. However, the programmer may change or alternate the radix interpretation by the use of the pseudo-operators DECIMAL and OCTAL.

The DECIMAL pseudo-op interprets all following numbers as decimal until the occurrence of the pseudo-op OCTAL.

The OCTAL pseudo-op resets the radix to its original octal base.

5.10.3 Extended Memory

The pseudo-op FIELD instructs the assembler to output a field setting so that it may recognize more than one memory field. This field setting is output during pass 2 and is recognized by the Run (or Load) command, which in turn causes all subsequent information to be loaded into the field specified by the expression. The form is:

FIELD n

n is an integer, a previously defined symbol, or an expression within the range 0<=n<=7.

This field setting is output on the binary file during pass 2 followed by an origin setting of 200. This word is read when the Run (or Load) command is executed and begins loading information into the new field.

The field setting is never remembered by the assembler and no initial field setting is output. A binary file produced without field settings will be loaded into field 0 when using the Run (or Load) command.

NOTE

A symbol in one field may be used to reference the same location in any other field. The field to which it refers is determined by the use of the CDF and CIF instructions. (The programmer who is unfamiliar with the IOT's but wishes to use them should refer to the PDP/8E SMALL COMPUTER HANDBOOK and experiment with several short test programs to satisfy himself as to their effect.)

CDF and CIF instructions must be used prior to any instruction referencing a location outside the current field, as shown in the following example:

```
          *200
          TAD P301
          CDF 00
          CIF 10
          JMS PRINT
          CIF 10
          JMP NEXT
P301,     301
          FIELD 1
          *200
NEXT,     TAD P302
          CDF 10
          JMS PRINT
          HLT
P302,     302
PRINT,    0
          TLS
          TSF
```

```
            JMP .-1
            CLA
            RDF
            TAD P6203
            DCA .+1
            000
            JMP I PRINT
P6203,      6203
```

When FIELD is used, the assembler follows the new FIELD setting with an origin at location 200. For this reason, if the programmer wants to assemble code at location 400 in field 1 he must write:

```
FIELD 1             /CORRECT EXAMPLE
*400
```

The following is incorrect and will not generate the desired code:

```
*400                /INCORRECT
FIELD 1
```

5.10.4 End-of-File

PAUSE signals the assembler to stop processing the file being read. The current pass is not terminated, and processing continues with the next file.

The PAUSE pseudo-op should be used only at the physical end of a file and with two or more segments of one program. When a PAUSE statement is reached the remainder of the file is ignored and processing continues with the next input file. PAUSE must be present or a PH error will occur.

5.10.5 Resetting the Location Counter

The PAGE n pseudo-op resets the location counter to the first address of page n, where n is an integer, a previously defined symbol, or a symbolic expression, all whose terms have been defined previously and whose value is from 0 to 37 inclusive, If n is not specified, the location counter is reset to the next logical page of memory. For example:

```
PAGE 2 sets the location counter to 00400
PAGE 6 sets the location counter to 01400
```

If the PAGE pseudo-op is used without an argument and the current location counter is at the first location of a page, it will not be moved. In the following example, the code TAD B is assembled into location 00400:

```
*377
            JMP .-3
PAGE
            TAD B
```

If several consecutive PAGE pseudo-ops are given, the first will cause the current location counter to be reset as specified. The rest of the PAGE pseudo-ops will be ignored.

5.10.6 Entering Text Strings

The TEXT pseudo-op allows a string of text characters to be entered as data and stored in 6-bit ASCII by using the pseudo-op TEXT followed by a space or spaces, a delimiting character (must be a printing character), the string of text, and the same delimiting character. Following the last character, a 6-bit zero is inserted as a stop code. For example:

```
TAG,        TEXT/123*/
```

The string would be stored as:

```
6162
6352
0000
```

5.10.7 Suppressing the Listing

Those portions of the source program enclosed by XLIST pseudo-ops will not appear in the listing file; the code will be assembled, however.

Two XLIST pseudo-ops may be used to enclose the code to be suppressed in which case the first XLIST with no argument will suppress the listing, and the second will allow it again. XLIST may also be used with an expression as an argument; a listing will be inhibited if the expression is equal to zero, or allowed if the expression is not equal to zero.

5.10.8 Reserving Memory

ZBLOCK instructs the assembler to reserve n words of memory containing zeros, starting at the word indicated by the current location counter. It is of the form:

```
ZBLOCK n
```

For example:

```
ZBLOCK 40
```

causes the assembler to reserve 40 (octal) words. The n may be an expression. If n=0, no locations are reserved.

5.10.9 Conditional Assembly Pseudo-Operators

The IFDEF pseudo-op takes the form:

IFDEF symbol <source code>

If the symbol indicated is previously defined, the code contained in the angle brackets is assembled; if the symbol is undefined, this code is ignored. Any number of statements or lines of code may be contained in the angle brackets. The format of the IFDEF statement requires a single space before and after the symbol.

The IFNDEF pseudo-op is similar in form to IFDEF and is expressed:

IFNDEF symbol <source code>

If the symbol indicated has not been previously defined, the source code in angle brackets is assembled. If the symbol is defined, the code in the angle brackets is ignored.

The IFZERO pseudo-op is of the form:

IFZERO expresson <source code>

If the evaluated (arithmetic or logical) expression is equal to zero, the code within the angle brackets is assembled; if the expression is non-zero, the code is ignored. Any number of statements or lines of code may be contained in the angle brackets. The expression may not contain any imbedded spaces and must have a single space preceding and following it.

IFNZRO is similar in form to the IFZERO pseudo-op and is expressed:

IFNZRO expression <source code>

If the evaluated (arithmetic or logical) expression is not equal to zero, the source code within the angle brackets is assembled; if the expression is equal to zero, this code is ignored.

Pseudo-ops can be nested, for example:

IFDEF SYM <IFNZRO X2 <...>>

The evaluation and subsequent inclusion or deletion of statements is done by evaluating the outermost pseudo-op first.

5.10.10 Controlling Binary Output

NOPUNCH causes the assembler to cease binary output but continue assembling code. It is ignored except during pass 2.

ENPUNCH causes the assembler to resume binary output after NOPUNCH, and is ignored except during pass 2. For example, these two pseudo-ops might be used where several programs share the same data on page zero. When these programs are to be loaded and executed together, only one page zero need be output.

5.10.11 Controlling Page Format

The EJECT pseudo-op causes the listing to jump to the top of the next page. A page eject is done automatically every 55 lines; EJECT is useful if the user requires more frequent paging. If this pseudo-op is followed by a string of characters, the first 40 (octal) characters of that string will be used as a new header line.

5.10.12 Altering the Permanent Symbol Table

PALC contains a table of symbol definitions for the PDP-8 and CAPS-8 peripheral devices. These are symbols such as TAD, DCA, and CLA, which are used in most PDP-8 programs. This table is considered to be the permanent symbol table for PALC; all of the symbols it contains are listed in Appendix C.

If the user purchases one or more optional devices whose instruction set is not defined among the permanent symbols (for example EAE or an A/D converter), he would want to add the necessary symbol definitions to the permanent symbol table in every program he assembles. Conversely, the user who needs more space for user-defined symbols would probably want to delete all definitions except the ones used in his program. For such purposes, PALC has three pseudo-ops that can be used to alter the permanent symbol table. These pseudo-ops are recognized by the assembler only during pass 1. During either pass 2 or pass 3 they are ignored and have no effect.

EXPUNGE deletes the entire permanent symbol table, except pseudo-ops.

FIXTAB appends all presently defined symbols to the permanent symbol table. All symbols defined before the occurrence of FIXTAB are made part of the permanent symbol table until the assembler is reloaded.

To append the following instructions to the symbol table, the user generates an ASCII file called SYMS.PAL containing:

```
MUY=7405        /MULTIPLY
DVI=7407        /DIVIDE
CLSK=6131       /SKIP ON CLOCK INTERRUPT
FIXTAB          /SO THAT THESE WON'T BE
                /PRINTED IN THE SYMBOL TABLE
```

The ASCII file is then entered in PALC's input designation. The user may also place the definitions at the beginning of the source file. This eliminates the need to load an extra file.

Each time the assembler is loaded, PALC's permanent symbol table is restored to contain only the permanent symbols shown in Appendix C.

The third pseudo-op used to alter the permanent symbol table in PALC is FIXMRI. FIXMRI is used to define a memory reference instruction and is of the form:

```
FIXMRI name=value
```

The letters FIXMRI must be followed by one space, the symbol for the instruction to be defined, an equal sign, and the value of the symbol. The symbol will be defined and stored in the symbol table as a memory reference instruction. The pseudo-op must be repeated for each memory reference instruction to be defined. For example:

```
EXPUNGE
FIXMRI  TAD=1000
FIXMRI  DCA=3000
CLA=7200
FIXTAB
```

When the preceding program segment is read into the assembler during pass 1, all symbol definitions are deleted and the three symbols listed are added to the permanent symbol table. Notice that CLA is not a memory reference instruction. This process can be performed to alter the assembler's symbol table so that it contains only those symbols used at a given installation or by a given program. This may increase the assembler's capacity for user-defined symbols in the program.

A summary of the PALC pseudo-ops is provided in Appendix C.

5.11 LINK GENERATION AND STORAGE

In addition to handling symbolic addressing on the current page of memory, PALC automatically generates links for off-page references. If reference is made to an address not on the page where an instruction is located, the assembler sets the indirect bit (bit 3) and an indirect address linkage will be generated on the current memory page. If the off-page reference is already an indirect one, the error diagnostic II (illegal indirect) will be generated. For example:

```
*2117
A,      CLA
        .
        .
        .
*2600
        JMP A
```

In the example above, the assembler will recognize that the register labelled A is not on the current page (in this case 2600 to 2777) and will generate a link to it as follows:

1. In location 2600 the assembler will place the word 5777 which is equivalent to JMP I 2777.

2. In address 2777 (the last available location on the current page) the assembler will place the word 2117 (the actual address of A).

During pass 3, the octal code for the instruction will be followed by an apostrophe (') to indicate that a link was generated.

Although the assembler will recognize and generate an indirect address linkage when necessary, the programmer may indicate an explicit

indirect address by the pseudo-op I. The assembler cannot generate a link for an instruction that is already specified as being as indirect reference. In this case, the assembler will print the error message II (illegal indirect). For example:

```
*2117
A,      CLA
        .
        .
        .
*2600
        JMP I A
```

The above coding will not work because A is not defined on the page where JMP I A is attempted, and the indirect bit is already set.

Literals and links are stored on each page starting at page address 177 (relative) and extending toward page address 0 (relative). Whenever the origin is then set to another page, the literal buffer for the current page is output. This does not affect later execution. There is room for 160 (octal) literals and links on page zero and 100 (octal) literals on each other page of memory.

Literals and links are stored only as far down as the highest instruction on the page. Further attempts to define literals will result in a PE (page exceeded) or ZE (page zero exceeded) error message.

5.12 CODING PRACTICES

A neat printout (or program listing, as it is usually called) makes subsequent editing, debugging, and interpretation much easier than if the coding were laid out in a haphazard fashion. The coding practices listed below are in general use, and will result in a readable, orderly listing.

1. A title comment begins with a slash at the left margin.

2. Pseudo-ops may begin at the left margin; often, however, they are indented one tab stop to line up with the executable instructions.

3. Address labels begin at the left margin. They are separated from succeeding fields by a tabulation.

4. Instructions, whether or not they are preceded by a label field, are indented one tab stop.

5. A comment is separated from the preceding field by one or two tabs (as required) and a slash; if the comment occupies the whole line itusually begins with a slash at the left margin.

5.13 PROGRAM PREPARATION AND ASSEMBLER OUTPUT

The following program was generated using the TAB function of the CAPS-8 EDITOR and was assembled with PALC.

```
*200
/EXAMPLE OF INPUT TO THE FORMAT
/GENERATOR PROGRAM
BEGIN,   0              /START OF PROGRAM
         KCC
         KSF            /WAIT FOR FLAG
         JMP .-1        /FLAG NOT SET YET
         KRB            /READ IN CHARACTER
         DCA CHAR
         TAD CHAR
         TAD MSPACE     /IS IT A SPACE?
         SNA CLA
         HLT            /YES
         JMP BEGIN+2    /NO:INPUT AGAIN
CHAR,    0              /TEMPORARY STORAGE
MSPACE,  -240
/END OF EXAMPLE
$
```

The program consists of statements and pseudo-ops and is terminated by the dollar sign ($). If the program is large, it can be segmented by placing it into several files; this often facilitates the editing of the source program since each section will be physically smaller.

The assembler initially sets the current location counter to 0200. This counter is reset whenever the asterisk (*) is processed.

The assembler reads the source file for pass 1 and defines all symbols used.

During pass 2, the assembler reads the source file and generates the binary code using the symbol table equivalences defined during pass 1. The binary file that is output may be loaded by the Load command. This binary file consists of an origin setting and data words.

During pass 3, the assembler reads the source file and generates the code from the source statements. The assembly listing is output in ASCII code. It consists of the current location counter, the generated code in octal, and the source statement. The 5-digit first column is the field number and 4-digit octal address (current location counter); the 4-digit second column is the assembled object code. The symbol table is printed at the end of the pass. The pass 3 output is:

```
*200                                    PALC-V1  03/08/73  PAGE 1

        0200  *200
              /EXAMPLE OF INPUT TO THE FORMAT
              /GENERATOR PROGRAM
00200   0000  BEGIN,  0              /START OF PROGRAM
00201   6032          KCC
00202   6031          KSF            /WAIT FOR FLAG
00203   5202          JMP .-1        /FLAG NOT SET YET
00204   6036          KRB            /READ IN CHARACTER
00205   3213          DCA CHAR
00206   1213          TAD CHAR
00207   1214          TAD MSPACE     /IS IT A SPACE?
00210   7650          SNA CLA
```

```
00211  7402           HLT              /YES
00212  5202           JMP BEGIN+2      /NO:INPUT AGAIN
00213  0000  CHAR,     0               /TEMPORARY STORAGE
00214  7540  MSPACE,  -240
             /END OF EXAMPLE
             $
```

```
*200                                  PALC-V1  03/08/73  PAGE 1-1

BEGIN   0200
CHAR    0213
MSPACE  0214
```

5.13.1 Terminating Assembly

PALC will a) terminate assembly, b) print a ↑C, and c) wait for the user to mount the System Cassette on drive 0 and type ↑C, under any of the following conditions:

1. Normal exit--The $ at the end of the source program was executed on pass 2 (or pass 3 if a listing is being generated).

2. Fatal error--One of the following error conditions was found and flagged (see the next section):

   BE DE DF PH SE

3. ↑C--If typed by the user, control turns to the Monitor.

5.14 PALC ERROR CONDITIONS

PALC will detect and flag error conditions and generate error messages on the console terminal. The format of the error message is:

CODE ADDRESS

where CODE is a 2-letter code which specifies the type of error, and ADDRESS is either the absolute octal address where the error occurred or the address of the error relative to the last symbolic tag (if there was one) on the current page. For example, the following code:

```
BEG,    TAD LBL
        %TAD LBL
```

would produce the error message:

```
IC BEG+0001
```

since % is an illegal character.

If at any time PALC prints ↑C, the user should make certain that the System Cassette is mounted on drive 0 and then type ↑C to return to the Monitor. He should examine each error indication to determine whether correction is required.

On the pass 3 listing, error messages are output as 2-character messages on the line just prior to the line in which the error occurred. The following table lists the PALC error codes. Those labeled Fatal Error are followed immediately by an effective ↑C.

Table 5-3 PALC Error Codes

Error Code	Explanation
BE	Two PAL-C internal tables have overlapped. Fatal error--assembly cannot continue.
DE	Device error. An error was detected when trying to read or write a device. Fatal error--assembly cannot continue.
DF	Device full. Fatal error--assembly cannot continue.
IC	Illegal character. The character is ignored and the assembly continued.
ID	Illegal redefinition of a symbol. An attempt was made to give a previously defined symbol a new value by means other than the equal sign. The symbol is not redefined.
IE	Illegal equals--an equal sign was used in the wrong context. Considered a warning and may not indicate an error but rather an undefined symbol at that point.
II	Illegal indirect--an off-page reference was made.
IP	Illegal pseudo-op--a pseudo-op was used in the wrong context or with incorrect syntax.
IZ	Illegal page zero reference--the pseudo-op Z was found in an instruction which did not refer to page zero. The Z is ignored.
PE	Current non-zero page exceeded--an attempt was made to:

Table 5-3 PALC Error Codes (Cont'd)

Error Code	Explanation
	1. Override a literal with an instruction 2. Override an instruction with a literal 3. Use more literals than the assembler allows on that page. This can be corrected by decreasing either the number of literals on the page or the number of instructions on the page.
PH	Phase error--either no $ appeared at the end of the program, or < and > in conditional pseudo-ops did not match. Fatal error--assembly cannot continue.
RD	Redefinition--a permanent symbol has been defined with =. The new and old definitions do not match. The redefinition is allowed.
SE	Symbol table exceeded--too many symbols have been defined for the amount of memory available. Fatal error--assembly cannot continue.
UO	Undefined origin--an undefined symbol has occurred in an origin statement.
US	Undefined symbol--a symbol has been processed during pass 2 that was not defined before the end of pass 1.
ZE	Page 0 exceeded--same as PE except with reference to page 0.

CHAPTER 6
CASSETTE BASIC

6.1 INTRODUCTION

Cassette BASIC is an interactive programming language derived from Dartmouth BASIC and designed to run under the Cassette Keyboard Monitor. The BASIC language is aimed at facilitating communication between the user and the computer. The user types his program as a series of numbered statements, making use of common English words and familiar mathematical notations. Because the BASIC language involves learning only a small number of commands, it is a very easy language to use. As the user gains familiarity with BASIC, he can add the advanced techniques available to perform more intricate manipulations or express a problem more efficiently and concisely.

Cassette BASIC provides approximately 1.7 to 2K of memory for program storage. Important features include 1- and 2-dimensional subscripting, user-coded functions, program chaining, use of cassettes for program storage, and use of line printer, if available, for output.

Beginning programmers may find a more fundamental approach to BASIC language programming in Chapter 1 of THE EDUSYSTEM HANDBOOK.

6.2 CALLING BASIC (.R BASIC)

Using the Cassette Keyboard Monitor, BASIC is called from the System Cassette by typing:

```
.R BASIC
```

When it is first loaded into memory, BASIC asks the user if he will use run-time file input and output as follows:

```
USING RUN-TIME FILE I/O?(Y OR N)
```

The user responds with Y or N followed by a carriage return. Choosing the run-time I/O feature leaves the user approximately 1.7K of memory for program storage, whereas a response of N frees the space used by the run-time I/O routines and provides an additional .3K of memory (enough for approximately 20-25 statements or 75 variables). Statements associated with the run-time I/O feature are:

```
OPEN...FOR INPUT
OPEN...FOR OUTPUT
CLOSE
IF END#
PRINT#
INPUT#
COMMAS
NO COMMAS
```

If any of these statements are used without the run-time I/O option having been chosen during BASIC's initial dialogue, BASIC will print a NO FILES ERROR message at run-time.

BASIC then asks:

```
NEW OR OLD-
```

The user responds NEW if he intends to create a program at the keyboard, and must respond with the name of the new program when BASIC requests:

```
NEW PROGRAM NAME-
```

The program name is typed as a standard system filename (6 characters or less) and an optional extension (1 to 3 characters); a program name is entered even if the user does not intend to save the program for future use. (A response of only a carriage return causes BASIC to repeat the NEW PROGRAM NAME request. If the user types an ALT MODE in response to this request, the name NONAME.BAS is assigned by BASIC.) When the new program name has been entered, BASIC indicates that it is ready to accept input by issuing a carriage return/line feed combination.

If the user responds OLD to BASIC's initial dialogue, BASIC assumes that the program has been previously saved on a cassette and will ask:

```
OLD PROGRAM NAME-
UNIT#(0-7):
```

The user must respond with the correct program name and file extension (if any), and then must specify which cassette unit drive the file is stored on. (An incorrect response will return an error message.) When this interaction is complete, BASIC will type:

```
READY.
```

and the user may edit or run his program.

6.3 NUMBERS

Cassette BASIC treats all numbers (in both integer and real formats) as real, or floating point, numbers. That is, BASIC accepts as input any number containing a decimal point and assumes a decimal point after any integer number entered.

In addition to integer and real formats, a third format is recognized and accepted by BASIC in order to express numbers outside the range .01<=x<1000000. This format is called exponential or E-type notation. In this format, a number is expressed as a decimal number times some power of 10, as follows:

where E represents "times 10 to the power of". A number in exponential notation is then read "xx times 10 to the power of n"; for example:

23.4E2 = 23.4*(10 to the power of 2) = 2340

Data may be input in any one or all three of these forms. Internal computations are carried out in floating point (real) format. Results of computations within the range .01<=x<1000000 are output as either real or integer decimal numbers (whichever is the correct but more concise format); results outside this range are output in exponential format. BASIC handles seven significant digits in normal operation and input/output, as illustrated below:

Value Typed In	Same Value Output By BASIC
.01	.01
.0099	9.900000E-3
999999	999999
1000000	1.000000E+6

BASIC automatically suppresses the printing of leading and trailing zeros in integer and decimal numbers and, as shown above, prints all exponential numbers in the form:

(sign) x.xxxxxx E (+ or -) n

where x represents the number carried to six decimal places, E stands for "times 10 to the power of", and n represents the exponent. For example:

-3.470218E+8 is equal to -347021800
7.260000E-4 is equal to .000726

6.4 VARIABLES

A variable in BASIC is a symbol which represents a number and is formed by a single letter or a letter followed by a digit. For example:

Acceptable Variables	Unacceptable Variables
I	2C - A digit cannot begin a variable
B3	AB - Two or more letters cannot form a variable
X	

The user may assign values to variables either by computing the values in a LET statement or by inputting the values as data; these operations are discussed later.

6.5 ARITHMETIC OPERATIONS

BASIC performs addition, subtraction, multiplication, division and exponentiation, as well as more complicated operations explained in detail later in the chapter. The five operators used in writing most formulas are:

Symbol	Meaning	Example
+	Addition	A + B
-	Subtraction	A - B
*	Multiplication	A * B
/	Division	A / B
↑	Exponentiation (Raise A to the Bth power)	A ↑ B

6.5.1 Priority of Operations

In any given mathematical formula, BASIC performs arithmetic operations in the following order of evaluation:

1. Parentheses receive top priority. Any expression within parentheses is evaluated before an unparenthesized expression.

2. In absence of parentheses, the order of priority is:

   a. Exponentiation

   b. Multiplication and Division (of equal priority)

   c. Addition and Subtraction (of equal priority)

3. If either 1 or 2 above does not clearly designate the order of priority, then the evaluation of expressions proceeds from left to right.

The expression A↑B↑C is evaluated from left to right as follows:

1. A↑B = step 1

2. (result of step 1)↑C = answer

The expression A/B*C is also evaluated from left to right since multiplication and division are of equal priority:

1. A/B = step 1

2. (result of step 1)*C = answer

6.5.2 Parentheses and Spaces

Parentheses may be used by the programmer to change the order of priority (as listed in rule 2 of the previous section). Since expressions within parentheses are always evaluated first, the programmer can control the order of evaluation by enclosing expressions appropriately. Parentheses may be nested, or enclosed by a second set (or more) of parentheses. In this case, the expression within the innermost parentheses is evaluated first, and then the next innermost, and so on, until all have been evaluated.

Consider the following example:

```
A=7*((B↑2+4)/X)
```

The order of priority is:

1. B↑2 = step 1
2. (result of step 1)+4 = step 2
3. (result of step 2)/X = step 3
4. (result of step 3)*7 = A

Parentheses also prevent any confusion or doubt as to how the expression is evaluated. For example:

```
A*B↑2/7+B/C+D↑2

((A*B↑2)/7)+((B/C)+D↑2)
```

Both of these formulas will be executed in the same way. However, most users will find that the second is easier to understand.

Spaces may be used in a similar manner. Since the BASIC compiler ignores spaces, the two statements:

```
LET B = D↑2 + 1
LETB=D↑2+1
```

are identical, but spaces in the first statement provide ease in reading.

6.5.3 Relational Operators

A program may require that two values be compared at some point to discover their relation to one another. To accomplish this, BASIC

makes use of the following relational operators:

=	equal to	>	greater than
<	less than	>=	greater than or equal to
<=	less than or equal to	<>	not equal to

Depending upon the result of the comparison, flow of program execution may be directed to another part of the program, or the validity of the relationship may cause a value of 0 (indicating a FALSE condition) or 1 (indicating a TRUE condition) to be assigned to a variable. For example:

```
15 X=Y<Z
```

This statement assigns the value 1 to X if Y is greater than Z. Relational operators are used primarily in conjunction with IF and LET statements, both of which are later discussed in detail.

The meaning of the equal sign (=) should be clarified. In algebraic notation, the formula X=X+1 is meaningless. However, in BASIC (and most computer languages), the equal sign designates replacement rather than equality. Thus, the formula X=X+1 is actually translated: "add one to the current value of X and store the new result back in the same variable X"; whatever value has previously been assigned to X will be combined with the value 1. An expression such as A=B+C instructs the computer to add the values of B and C and store the result in a third variable A; the variable A is not being evaluated in terms of any previously assigned value, but only in terms of B and C. Therefore, if A has been assigned any value prior to its use in this statement, the old value is lost; it is instead replaced by the value of B+C. Finally, the equal sign may be used in relational testing as illustrated in the previous example.

6.6 IMMEDIATE MODE

Commands are available which allow Cassette BASIC to act as a calculator--that is, the user types an algebraic expression which is to be calculated and BASIC types back the result. This is called Immediate Mode since the user is not required to write a detailed program to calculate expressions and equations, but can use BASIC to produce results immediately. The commands used in Immediate Mode are PRINT, LET and occasionally the FOR-NEXT combination. These are explained in the following paragraphs.

6.6.1 PRINT Command

The PRINT command is of the form:

```
PRINT expression
```

BASIC is instructed to compute the value of the expression and print the result on the console terminal. The expression is a normal

arithmetic expression which may include numbers, variables, arithmetic operators, and functions (discussed in Section 6.8.12). A string of text may also be printed (see Section 6.8.5--PRINT). For example:

```
PRINT 1/8↑8
 5.960464E-Ø8
```

6.6.2 LET Command

Values may be assigned to variables by use of the LET command as follows:

LET variable=expression

The computer does not type anything in response to this command, but computes the expression and assigns the value to the variable. The variable may then be used in another computation or may be output using the PRINT command. For example:

```
LET P1=3.14159

PRINT P1*4↑2
 5Ø.26544
```

6.6.3 Looping PRINT and LET Commands

It is possible to include PRINT and LET commands in a loop so that variables and results may be stored or printed in a series. Looping is accomplished by means of FOR-NEXT statements in which the FOR statement sets the limits of the loop and the NEXT statement increments the count by 1. The only restriction in Immediate Mode looping is that the command and the looping statements must appear on one line. This is accomplished by using the backslash (\) character to separate multiple statements on a line. (The backslash is produced on an LT33 or 35 Teletype by pressing the SHIFT and L keys simultaneously. Other types of terminals provide a separate key.) For example:

```
LET P1=3.14159
FOR I=1 TO 3\PRINT P1↑I\NEXT I
```

This combination will print the results of 3.14159 to the 1st, 2nd, and 3rd powers respectively.

More information on looping in general is provided in Section 6.8.7.

6.7 EXAMPLE RUN

The following Example Run is included at this point as an illustration of Cassette BASIC's initial dialogue, the format of a BASIC program, the ease in editing and running it, and the type of output that may be produced. The user calls in the program AVER from cassette drive 1 and attempts to run it. Execution is halted by a SYNTAX ERROR at line 30. The user lists the program, finds the mistake in line 30, and also notices a mistake in line 85. He corrects these errors by retyping the lines, and then reruns the program. After execution he saves the corrected program on drive 1 under the original name.

Following sections cover the statements and commands used in BASIC programming.

```
.R BASIC
USING RUN-TIME FILE I/O?(Y OR N)N

NEW OR OLD-OLD

OLD PROGRAM NAME-AVER
UNIT#(Ø-7):1


READY.


RUN
HOW MANY STUDENTS, HOW MANY GRADES PER STUDENT ?5,4
SYNTAX ERROR AT LINE 3Ø

LIST
1Ø REM - PROGRAM TO TAKE AVERAGE OF
15 REM - STUDENT GRADES AND CLASS GRADES
2Ø PRINT "HOW MANY STUDENTS, HOW MANY GRADES PER STUDENT ";
3Ø INPUT A,1B
4Ø LET I=1
5Ø FOR J=I TO A
55 LET V=Ø
6Ø PRINT "STUDENT NUMBER =";J
75 PRINT "ENTER GRADES"
76 LET D=J
8Ø FOR K=D TO D+(B-1)
81 INPUT G
82 LET V=V+G
85 NEXT L
9Ø LET V=V/B
95 PRINT "AVERAGE GRADE =";V
96 PRINT
99 LET Q=Q+V
1ØØ NEXT J
1Ø1 PRINT
1Ø2 PRINT
1Ø3 PRINT "CLASS AVERAGE =";Q/A
1Ø4 STOP
14Ø END
```

```
READY.


30 INPUT A,B
85 NEXT K
RUN
HOW MANY STUDENTS, HOW MANY GRADES PER STUDENT ?5,4
STUDENT NUMBER = 1
ENTER GRADES
?78
?86
?88
?74
AVERAGE GRADE = 81.5

STUDENT NUMBER = 2
ENTER GRADES
?59
?86
?70
?87
AVERAGE GRADE = 75.5

STUDENT NUMBER = 3
ENTER GRADES
?58
?64
?75
?80
AVERAGE GRADE = 69.25

STUDENT NUMBER = 4
ENTER GRADES
?88
?92
?85
?79
AVERAGE GRADE = 86

STUDENT NUMBER = 5
ENTER GRADES
?60
?78
?85
?80
AVERAGE GRADE = 75.75


CLASS AVERAGE = 77.6

READY.


SAVE

UNIT#(0-7):1


READY.
```

6.8 BASIC STATEMENTS

The statements described in this section are used in creating BASIC programs. These statements make up the body of the program; they perform arithmetic calculations and input and output operations, and control the order of program execution.

6.8.1 Statement Numbers

An integer number is placed at the beginning of each line in a BASIC program. BASIC executes the statements in a program in numerically consecutive order regardless of the order in which they have been typed. A recommended practice is to number lines by fives or tens, so that additional lines may be inserted in a program without the necessity of renumbering lines already present. (BASIC programs may be created using either the BASIC Editor as described here, or the CAPS-8 EDITOR. If the CAPS-8 EDITOR is used, the programmer must make certain to type his program in numerically consecutive order, as BASIC will not sort it in this case.)

Multiple statements may be placed on a single line by separating each statement from the preceding statement with a backslash (SHIFT/L). This feature is particularly useful since statement numbers require space in the symbol table; if unnecessary statement numbers are eliminated by use of the backslash, there will be more room for program storage. For example:

```
10 A=5\B=.2\C=3\PRINT "ENTER DATA"
```

All of the statements in line 10 will be executed before BASIC continues to the next line. Only one statement number at the beginning of the entire line is necessary. However, it should be noted that program control cannot be transferred to a statement within a line, but only to the first statement of the line in which it is contained (see Section 6.8.9, Transfer of Control Statements).

6.8.2 Commenting the Program (REM)

The REM or REMARK statement allows the programmer to insert comments or remarks into a program without these comments affecting execution. The BASIC compiler ignores everything on a line beginning with REM. The form is:

(line number) REM (message)

In the Example Run program, lines 10 and 15 are REMARK statements describing what the program does. It is often useful to put the name of the program and information relating to its use at the beginning where it is available for future reference. Remarks throughout the body of a long program will help later debugging by explaining the purpose of each section of code within the program.

6.8.3 Terminating the Program (END and STOP)

The END statement (line 140 in the Example Run program), if present, must be the last statement of the entire program. The form is:

(line number) END

Use of the END statement is optional. If executed, it signals the end of the program and BASIC prints:

```
READY.
```

Variables and arrays are left in an undefined state, thereby losing any values they have been assigned during execution.

The STOP statement is used synonymously with the END statement to terminate execution, but while END occurs only once at the end of a program, STOP may occur any number of times. The format of the STOP statement is:

(line number) STOP

This statement signals that execution is to be terminated at that point in the program where it is encountered leaving variables in a defined state. (Variables will contain the values assigned when the statement is encountered.)

6.8.4 The Arithmetic Statement (LET)

The Arithmetic (LET) statement is probably the most commonly used BASIC statement. It causes a value to be assigned to a variable and is of the form:

(line number) (LET) x = expression

where x represents a variable, and the expression is either a number, another variable, or an arithmetic expression. The word 'LET' is optional; thus the following statements are treated the same:

```
LET A=A↑B+10          LET C=F/G
A=A↑B+10              C=F/G
```

As mentioned earlier, relational operators may be used in a LET statement to assign a value to a variable depending upon the validity of a relationship. If the statement is FALSE, the value 0 is assigned to the variable; if TRUE, the value 1 is assigned. For example:

```
100 A=1
105 B=2
110 C=A=B
120 D=A>B
130 E=A<>B
140 PRINT C,D,E
150 END
```

Translated, this actually means "let C=1 if A=B (0 otherwise); let D=1 if A>B (0 otherwise)" and so on. Thus, the values of C, D, and E are printed as follows:

```
RUN
 Ø              Ø              1

READY.
```

There is no limit to the number of relationships that may be tested in the statement.

6.8.5 Input/Output Statements

Input/Output statements allow the user to bring data into a program and output results or data at any time during execution. The console terminal keyboard, (LT33 Teletype reader and punch units, if present), cassettes, and line printer are all available as I/O devices in Cassette BASIC. Statements which control their use are described next.

READ and DATA

READ and DATA statements are used to input data into a program. One statement is never used without the other. The form of the READ statement is:

(line number) READ x1,x2,...xn

where x1 through xn represent variable names. For example:

```
1Ø READ A,B,C
```

A,B, and C are variables to which values will be assigned. Variables in a READ statement must be separated by commas. READ statements are generally placed at the beginning of a program, but must at least logically occur before that point in the program where the value is required for some computation.

Values which will be assigned to the variables in a READ statement are supplied in a DATA statement of the form:

(line number) DATA x1,x2,...xn

where x1 through xn represent values. The values must be separated by commas and must occur in the same order as the variables which are listed in the corresponding READ statement. A DATA statement appropriate for the preceding READ statement is:

```
7Ø DATA 1,2,3
```

Thus, after executing the READ statement, A=1, B=2, and C=3.

The DATA statement is usually placed at the end of a program (before the END statement) where it is easily accessible to the programmer should he wish to change the values.

A READ statement may have more or fewer variables than there are values in any one DATA statement. The READ statment causes BASIC to search all available DATA statements in consecutive line number order until values are found for each variable in the READ. A second READ statement will begin reading values where the first stopped. If at some point in the program an attempt is made to read data which is not present or if the data is not separated by commas, BASIC will stop and print the following message on the console terminal:

DATA ERROR AT LINE XXXX

where XXXX indicates the line number of the READ statement which caused the error.

RESTORE

If it should become necessary to use the same data more than once in a program, the RESTORE statement will make it possible to recycle through the DATA statements beginning with the lowest numbered DATA statement. The RESTORE statement is of the form:

(line number) RESTORE

An example of its use follows:

```
15 READ B,C,D
.
.
.
55 RESTORE
60 READ E,F,G
.
.
.
80 DATA 6,3,4,7,9,2
.
.
100 END
```

In this example, the READ statements in lines 15 and 60 will both read the first three data values provided in line 80. If the RESTORE statement had not been inserted before line 60, then the second READ would pick up data in line 80 starting with the fourth value.

In recycling through data with a RESTORE statement, the programmer may use the same variable names the second time through the data, or not, as he chooses, since the values are being read as though for the first time. In order to skip unwanted values, the programmer may insert replacement (or dummy) variables. Consider:

```
1 REM - PROGRAM TO ILLUSTRATE USE OF RESTORE
20 READ N
25 PRINT "VALUES OF X ARE:"
30 FOR I=1 TO N
40 READ X
50 PRINT X,
60 NEXT I
70 RESTORE
185 PRINT
190 PRINT "SECOND LIST OF X VALUES"
200 PRINT "FOLLOWING RESTORE STATEMENT:"
210 FOR I=1 TO N
220 READ X
230 PRINT X,
240 NEXT I
250 DATA 4,1,2
251 DATA 3,4
300 END

RUN
VALUES OF X ARE:
 1             2             3             4
SECOND LIST OF X VALUES
FOLLOWING RESTORE STATEMENT:
 4             1             2             3
READY.
```

The second time the data values are read, the variable X (line 220) picks up the value originally assigned to N in line 20, and as a result, BASIC prints:

```
4             1             2             3
```

To circumvent this, the programmer could insert a dummy variable (for example, 205 READ Z), which would pick up and store the first value, but would not be represented in the PRINT statement. In this case the output would be the same each time through the list.

INPUT

The INPUT statement is used when data is to be supplied by the user from the console terminal keyboard while a program is executing, and is of the form:

(line number) INPUT x1,x2,...xn

where x1 through xn represent variable names. For example:

```
25 INPUT A,B,C
```

This statement will cause the program to pause during execution, print a question mark on the console terminal, and wait for the user to type three numerical values. The user must separate the values by commas; they are entered into the computer by pressing the RETURN key at the end of the list.

If the user does not insert enough values to satisfy the INPUT statement, BASIC prints another question mark and waits for more values to be input. When the correct number has been entered, execution continues. If too many values are input, BASIC ignores those in excess of the required number. The values are entered only when the user types the RETURN key.

OPEN

Input and output files may be stored on cassette, and may be accessed during run-time (providing the user has chosen the run-time I/O option during BASIC's initial loading dialogue). Before an I/O file is accessed however, the user must first open it via one of the following commands:

(line number) OPEN "n:xxxx" FOR INPUT

or

(line number) OPEN "n:xxxx" FOR OUTPUT

where n represents the cassette drive number (0-7), and xxxx is any legal filename (6 characters or less, and optional extension of 3 characters or less). Input files are created either by using BASIC or the CAPS-8 EDITOR (see Section 6.8.6), and must have been previously stored on cassette before being accessed. For example, the statement:

```
215 OPEN "1:TEST.DAT" FOR INPUT
```

opens an input file named TEST.DAT on cassette drive 1.

Only one input and one output file may be open at any time, and only one file--either input or output--may be open on a given cassette drive at one time.

CLOSE

The CLOSE statement is used to close a currently open output file, and is of the form:

(line number) CLOSE

Suceeding OPEN FOR INPUT statements will perform an automatic close on a previously open input file; however, the user should take note of the following cases:

1. If the user attempts to open an input file on a cassette which is currently open for output, BASIC will return an I O ERROR, as the same cassette drive cannot be open for both input and output at the same time.

2. If the user has an input file open on a cassette, and is at its end-of-file (that is, a CTRL/Z has been detected), BASIC will allow him to open an output file

on the same cassette, since the input file is theoretically "closed". However, if the user has an input file open on a cassette and is not at its end-of-file, an I O ERROR will occur if he then tries to open an output file on the same cassette. (See Section 6.8.9, IF END#, for more information on BASIC's method of detecting an end-of-file.)

3. If the user tries to open an output file and an output file is already open on any cassette, BASIC will return a "FILE OPEN ERROR"; before opening a new output file, the current output file must be closed.

A close is automatically performed on both open input and open output files by STOP, END and CHAIN statements, as well as by all errors detected at run-time.

INPUT#

Once an input file has been opened using the open statement, data can be called into a program using the INPUT# statement. The form of this statement is:

(line number) INPUT# x1,x2,...xn

where # signifies that the file is stored on cassette under the filename and drive number specified in the last "OPEN...FOR INPUT" statement; x1 through xn represent variable names.

When the BASIC program reaches the INPUT# statement during execution, the data is automatically called into the program from cassette and execution continues. INPUT# statements and INPUT statements may be interspersed throughout a program. The input file need only be opened once before it is referenced.

PRINT

The PRINT statement is used to output results of computations, comments, values of variables, or plot points of a graph on the console terminal. The format is:

(line number) PRINT expression

When no expression is indicated in the statement line, a blank line is output. For example:

```
205 PRINT
210 PRINT
```

Two blank lines will be output on the console terminal. By using certain kinds of expressions and the control characters colon and semicolon, the user can create fairly sophisticated formats.

In order to print out the results of a computation and the value of a variable, the user types the line number, PRINT, and the variable name(s) separated by a format control character (in this case, commas) as follows:

```
5 A=16\B=5\C=4
10 PRINT A,C+B,SQR(A)
```

In BASIC, an output line is formatted into five columns (called print zones) of 14 spaces each. The control character comma causes a value to be typed beginning at the next available print zone. In the above example, the value of A, the sum of A+B, and the square root of A are printed in the first three print zones as follows:

```
RUN
 16             9              4
```

A statement such as in line number 10 in this next example:

```
5 A=2.3\B=21\C=156.75\D=1.134\E=23.4
10 PRINT A,B,C,D,E
```

causes the values of the variables to be printed in the same format using all five zones:

```
RUN
 2.3            21             156.75         1.134          23.4
```

When more than five variables are listed in the PRINT statement, the sixth value begins a new line of output.

The PRINT statement may also be used to output a message or line of text. The desired message is simply placed in quotation marks in the PRINT statement as follows:

```
10 PRINT "THIS IS A TEST"
```

when line 10 is encountered during execution, the following is printed:

```
THIS IS A TEST
```

A message may be combined with the result of a calculation or a variable as follows:

```
80 PRINT "AMOUNT PER PAYMENT =",R
```

Assuming R=344.96, when line 80 is encountered during execution, the results are output as:

```
 AMOUNT PER PAYMENT =          344.96
```

If a number following a printed message is too long to be printed on a single line, the number is automatically moved to the beginning of the next line.

It is not necessary to use the standard 5-zone format for output. The control character semicolon (;) causes the text or data to be output

immediately after the last character printed (separated from that character by a space and followed by another space). If neither a comma nor a semicolon is used, BASIC assumes a semicolon. Thus both of the following:

```
80 PRINT "AMOUNT PER PAYMENT ="R
80 PRINT "AMOUNT PER PAYMENT =";R
```

result in:

```
AMOUNT PER PAYMENT = 344.96
```

The PRINT statement can also cause a constant to be printed on the console terminal. (This is similar to the PRINT command used in Immediate Mode.) For example:

```
10 PRINT 1.234,SQR(10014)
```

causes the following to be output at execution time:

```
1.234          100.07
```

Any algebraic expression in a PRINT statement is evaluated using the current value of the variables. Numbers are printed according to the format discussed in Section 6.3.

The following example program illustrates the use of the control characters comma and semicolon in PRINT statements. The user may also wish to refer to Section 6.8.12 for information pertaining to three functions available for additional character control--TAB, PUT, and GET:

```
10 READ A,B,C
20 PRINT A,B,C,A↑2,B↑2,C↑2
30 PRINT
40 PRINT A;B;C;A↑2;B↑2;C↑2
50 DATA 4,5,6
60 END

RUN
 4             5             6             16            25
 36

 4  5  6  16  25  36

READY.
```

Another use of the PRINT statement is to combine it with an INPUT statement so as to identify the data expected to be entered. As an example, consider the following program:

```
10 REM - PROGRAM TO COMPUTE INTEREST PAYMENTS
20 PRINT "INTEREST IN PERCENT";
25 INPUT J
26 LET J=J/100
30 PRINT "AMOUNT OF LOAN";
35 INPUT A
40 PRINT "NUMBER OF YEARS";
45 INPUT N
50 PRINT "NUMBER OF PAYMENTS PER YEAR";
55 INPUT M
60 LET N=N*M
65 LET I=J/M
70 LET B=1+I
75 LET R=A*I/(1-1/B↑N)
78 PRINT
80 PRINT "AMOUNT PER PAYMENT =";R
85 PRINT "TOTAL INTEREST      =";R*N-A
88 PRINT
90 LET B=A
95 PRINT " INTEREST      APP TO PRIN      BALANCE"
100 LET L=B*I
110 LET P=R-L
120 LET B=B-P
130 PRINT L,P,B
140 IF B>=RGO TO 100
150 PRINT B*I,R-B*I
160 PRINT "LAST PAYMENT ="B*I+B
200 END
RUN
INTEREST IN PERCENT?9
AMOUNT OF LOAN?2500
NUMBER OF YEARS?2
NUMBER OF PAYMENTS PER YEAR?4

AMOUNT PER PAYMENT = 344.9617
TOTAL INTEREST     = 259.6932

 INTEREST      APP TO PRIN      BALANCE
 56.25          288.7117        2211.288
 49.75399       295.2077        1916.081
 43.11182       301.8498        1614.231
 36.32019       308.6415        1305.589
 29.37576       315.5859        990.0035
 22.27508       322.6866        667.317
 15.01463       329.947         337.3699
 7.590824       337.3708
LAST PAYMENT = 344.9608

READY.
```

As can be noticed in this example, the question mark is grammatically useful when several values are to be input by allowing the programmer to formulate a verbal question which the input values will answer.

PRINT#

The PRINT# statement is similar to the PRINT statement with the exception that data and messages are sent to the current output file on cassette rather than to the console terminal. The form of the statement is:

(line number) PRINT# x1,x2,...xn

where # signifies that the output will be sent to the cassette drive number and filename of the currently open output file, and x1 through xn represent data variables. (The current open file is determined by the OPEN FOR OUTPUT statement, as detailed earlier in this section.)

If the user attempts to save data on a full cassette, BASIC prints an error message and returns control to its editing phase. The data already output is lost, and the user will have to rerun his program using a different output cassette.

COMMAS and NO COMMAS

Data stored in an output file on cassette is often called later as input by another or the same program. (This is in fact the only method of passing data between segments of a chained program.) In order to be used as input, this data must be in the same format as it would appear if written in a DATA statement. Cassette BASIC provides two statements for formatting this output--COMMAS and NO COMMAS.

In order to be used as data, individual values must be separated by commas; the COMMAS statement inserts a comma after each item of data; (unless the COMMAS statement is inserted in the program prior to PRINT# statement, data will be output in the format illustrated earlier under the PRINT statement.) The form is:

(line number) COMMAS

A NO COMMAS statement will set the format back to its original state. The COMMAS and NO COMMAS statements do not affect output on either the console terminal or line printer.

The following example writes out four values in a file called "OUT.DAT", reads the values back into memory and prints them on the console terminal.

```
10 OPEN "1:OUT.DAT" FOR OUTPUT
15 COMMAS
20 PRINT# 1;2;3;4
30 CLOSE
40 OPEN "1:OUT.DAT" FOR INPUT
50 INPUT# I,J,K,L
60 PRINT I,J,K,L
70 END
```

Output appears as follows:

```
RUN
 1            2            3            4

READY.
```

The COMMAS statement is not necessary if the user is only sending one value per line. The preceding example could have been coded as follows, with the same results:

```
10 OPEN "1:OUT.DAT" FOR OUTPUT
20 FOR I=1 TO 4
30 PRINT# I
40 NEXT I
50 CLOSE
60 OPEN "1:OUT.DAT" FOR INPUT
70 INPUT# I,J,K,L
80 PRINT I,J,K,L
90 END
```

In this case the file OUT.DAT would appear:

```
1
2
3
4
```

whereas in the first case it would appear as follows:

```
1,2,3,4
```

The user must take care when inputting data from cassette files. For example, if the file OUT.DAT is in the form:

```
1,2,3,4
```

and the user attempts to input these values using the following statement:

```
50 INPUT# I,J,K
```

the proper values for I, J, and K will be read, but the rest of the line will be lost as far as satisfying any future variables--just as it would be lost if these values were input from the console terminal. (Refer to the information concerning the INPUT statement in this section.)

LPT

The LPT statement is used to generate output on the line printer (if one is available) and is of the form:

(line number) LPT

By inserting this statement anywhere in a program, all subsequent output, with the exception of error messages, will be printed on the line printer. The LPT statement is particularly advantageous for outputting large amounts of calculated data, as can be seen from this

and following examples:

```
100 LPT
110 FOR F=30 TO 60 STEP 3
120 PRINT F,F↑2
130 NEXT F
140 END

RUN

30            900
33            1089
36            1296
39            1521
42            1764
45            2025
48            2304
51            2601
54            2916
57            3249
60            3600
```

When the END statement is encountered in the program, the output device is reset to the console terminal.

TTY OUT

The console terminal may be placed under program control so that during execution of a program output may be sent alternately between the console terminal and the line printer (if one is available on the system).

Control is originally set with the console terminal. By issuing the LPT statement discussed previously, all subsequent output can be sent to the line printer. To return control to the console terminal from within the program, the statement:

(line number) TTY OUT

is inserted. (Cassette I/O always returns control to the last device indicated, so that the TTY OUT statement need only be used when the line printer is involved.)

The following program makes use of almost all the available I/O devices. The console terminal and line printer output is included.

```
5 REM PROGRAM TO DEMONSTRATE ALL I/O DEVICES
10 REM AVAILABLE IN CASSETTE BASIC
15 REM
20 PRINT "PROGRAM TO CALCULATE SQUARES AND SQUARE ROOTS"
25 PRINT
27 REM GET LOOP LIMITS FROM USER
30 PRINT "INPUT LOWER LIMIT"
35 INPUT L
40 PRINT "INPUT UPPER LIMIT"
45 INPUT U
50 PRINT "INPUT STEP"
55 INPUT S
57 REM CREATE A CASSETTE FILE OF SQUARES OF NUMBERS
60 OPEN "1:SQUARE.DAT"FOR OUTPUT
```

```
65 LPT
66 REM PRINT A FORM FEED ON LINEPRINTER
70 T=PUT(12)
75 PRINT "TABLE OF NUMBERS AND THEIR SQUARES"
80 PRINT
81 PRINT
82 PRINT " X"," X↑2"
83 PRINT
85 FOR X=L TO J STEP S
90 PRINT X,X↑2
95 REM ALSO SEND SQUARES TO CASSETTE FILE
100 PRINT# X↑2
105 NEXT X
106 CLOSE
110 T=PUT(12)
111 TTY OUT
112 PRINT "TABLE OF SQUARES COMPLETE"
113 LPT
115 PRINT "TABLE OF NUMBERS AND THEIR SQUARE ROOTS"
120 OPEN "1:SQUARE.DAT"FOR INPUT
125 PRINT
126 PRINT
127 PRINT " X"," SQR(X)"
128 PRINT
130 FOR X=L TO J STEP S
135 INPUT# J
136 PRINT J,SQR(J)
140 NEXT X
150 T=PUT(12)
155 TTY OUT
160 PRINT "PROGRAM COMPLETED"
165 END
RUN
PROGRAM TO CALCULATE SQUARES AND SQUARE ROOTS

INPUT LOWER LIMIT
?1
INPUT UPPER LIMIT
?50
INPUT STEP
?1
TABLE OF SQUARES COMPLETE
```

TABLE OF NUMBERS AND THEIR SQUARES

X	X^2
1	1
2	4
3	9
4	16
5	25
6	36
7	49
8	64
9	81
10	100
.	
.	
49	2401
50	2500

TABLE OF NUMBERS AND THEIR SQUARE ROOTS

X	SQR(X)
1	1
4	2
9	3
16	4
25	5
36	6
49	7
64	8
81	9
100	10
.	
.	
2401	49
2500	50

NOTE

If an LT33 Teletype is used as the console terminal and it includes a reader and punch, these devices may be used for I/O operations at any time; no special statement is required. To read in data from the reader, position the tape over the sprocket wheel; when input is required, set the reader to START and the tape will begin reading in. To punch a tape, set the punch to ON and all Teletype output will be punched on the punch. Using the paper tape I/O devices is, in effect, the same as using the Teletype keyboard. Characters will be typed on the Teletype keyboard as tapes are being read or punched.

6.8.6 Creating Run-Time Input Files

Data files stored on cassette and used for input during execution can be created either by use of BASIC itself or by use of the CAPS-8 EDITOR.

Using BASIC, the programmer creates a program which accepts values from the console terminal keyboard and then writes these onto the cassette as an output file. Data files consist of consecutive ASCII characters. If the useful data in a file is to end before the actual end-of-file, the last useful character must be followed by a CTRL/Z. (This character is inserted by BASIC when the user closes an output file. When later detected during input, BASIC sets an end-of-file flag; the user can test an end-of-file condition by using the IF-END# statement.) The COMMAS statement is used to produce the correct format for a data file when more than one value is on a single line.

The following program illustrates one method of doing this:

```
5 REM - PROGRAM TO ACCEPT DATA FROM THE CONSOLE
10 REM - TERMINAL AND CREATE A RUNTIME INPUT FILE
20 OPEN "0:RTIN.DAT" FOR OUTPUT
25 PRINT "INPUT A,B,C,D";
30 INPUT A,B,C,D
35 COMMAS
40 PRINT# A,B,C,D
45 PRINT "INPUT F(I) FOR I=1 TO 10"
50 DIM F(10)
52 REM - COMMAS NOT NEEDED SINCE ARRAY WILL
53 REM - BE OUTPUT ONE ELEMENT PER LINE
55 NO COMMAS
60 FOR I=1 TO 10
70 PRINT "F("I")";
75 INPUT F(I)
80 PRINT# F(I)
85 NEXT I
90 PRINT "INPUT V1,V2,Z"
95 INPUT V1,V2,Z
97 REM - COMMAS ARE NEEDED SINCE V1, V2 AND Z
98 REM - WILL BE OUTPUT ON THE SAME LINE
```

```
100 COMMAS
105 PRINT# V1,V2,Z
110 CLOSE
115 END
    RUN
    SAVE

    UNIT#(0-7):0



    READY.
```

The CAPS-8 EDITOR can also be used to create an input file. The EDITOR first asks for input and output devices and filenames; then the user types the file using EDITOR commands and making sure the format is correct for BASIC. The same data file in the above example can be created using the EDITOR as follows:

```
.R EDIT
*INPUT FILE-
*OUTPUT FILE-0:MTIN.DAT

#A
1.37,2.346,-13.267,-1.056
23
3.56
1.436
38
9.026
23.067
89
54
12.467
-1
123,34567,789

#L
1.37,2.346,-13.267,-1.056
23
3.56
1.436
38
9.026
23.067
89
54
12.467
-1
123,34567,789

#E

.
```

6.8.7 Loops (FOR, NEXT and STEP)

A loop is a set of instructions which are repeated over and over again, each time being modified in some way until a terminal condition is reached. FOR and NEXT statements define the beginning and end of a loop; STEP specifies an incremental value. The FOR statement is of the form:

(line number) FOR v=x1 TO x2 STEP x3

where v represents a variable name, and x1, x2, and x3 all represent formulas (a formula in this case means a numerical value, variable name, or mathematical expression). v is termed the index, x1 the initial value, x2 the terminal value, and x3 the incremental value. For example:

```
15 FOR K=2 TO 20 STEP 2
```

This loop will be repeated as long as K is less than or equal to 20. Each time through the loop, K is incremented by 2, so the loop will be executed a total of 10 times.

A variable used as an index in a FOR statement must not be subscripted, although a common use of loops is to deal with subscripted variables using the value of the index as the subscript of a previously defined variable (this is illustrated in Section 6.8.8, Subscripted Variables).

The NEXT statement is of the form:

(line number) NEXT v

where v is the index of the FOR loop and signals the end of the loop. When execution of the loop reaches the NEXT statement, the computer adds the STEP value to the index and checks to see if the index is less than or equal to the terminal value. If so, the loop is executed again. If the value of the index exceeds the terminal value, control falls through the loop to the following statement, with the value of the index equaling the value it was assigned the final time through the loop. (Note that this method of handling loops varies among other versions of the BASIC language.)

If the STEP value is omitted, a value of +1 is assumed. (Since +1 is the usual STEP value, that portion of the statement is frequently omitted.) The STEP value may also be a negative number.

The following example illustrates the use of loops. This loop is executed 10 times: the value of I is 10 when control leaves the loop. +1 is the assumed STEP value.

```
10 FOR I=1 TO 10
20 NEXT I
30 PRINT I
40 END

RUN
 10

READY.
```

If line 10 had been:

```
10 FOR I=10 TO 1 STEP -1
```

the value printed by the computer would be 1.

As indicated earlier, the numbers used in the FOR statement are formulas; these formulas are evaluated upon first encountering the loop. While the index, initial, terminal and STEP values may be changed within the loop, the value assigned to the initial formula remains as originally defined until the terminal condition is reached. To illustrate this point, consider the previous example. The value of I (in line 10) can be successfully changed as follows:

```
10 FOR I=1 TO 10
15 LET I=10
20 NEXT I
```

The loop will only be executed once since the value 10 has been reached by the variable I and the terminal condition is satisfied.

If the value of the counter variable is originally set equal to the terminal value, the loop will execute once, regardless of the STEP value. If the starting value is beyond the terminal value, the loop will also execute only once.

It is possible to exit from a FOR-NEXT loop without the index reaching the terminal value. This is known as a conditional transfer and is explained in Section 6.8.9. Control may only transfer into a loop which has been left earlier without being completed, ensuring that the terminal and STEP values are assigned.

Nesting Loops

It is often useful to have one or more loops within a loop. This technique is called nesting, and is allowed as long as the field of one loop (the numbered lines from the FOR statement to the corresponding NEXT statement, inclusive) does not cross the field of another loop. A diagram is the best way to illustrate acceptable nesting procedures:

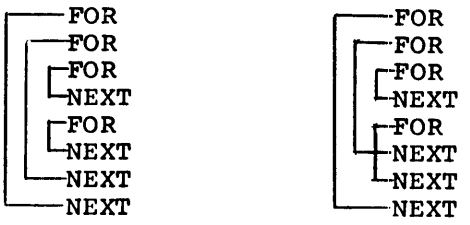
ACCEPTABLE NESTING TECHNIQUES	UNACCEPTABLE NESTING TECHNIQUES

Two Level Nesting

```
┌──FOR
│ ┌FOR
│ └NEXT
│ ┌FOR
│ └NEXT
└──NEXT
```

```
┌─FOR
│ ┌FOR
│ └─NEXT
└──NEXT
```

Three Level Nesting



A maximum of eight (8) levels of nesting is permitted. Exceeding that limit will result in the error message:

FOR ERROR AT LINE XXXX

where XXXX is the number of the line in which the error occurred.

6.8.8 Subscripted Variables

In addition to single variable names, BASIC accepts another class of variables called subscripted variables. Subscripted variables provide the programmer with additonal computing capabilities for handling lists, tables, matrices, or any set of related variables. Variables are allowed one or two subscripts. A single letter forms the name of the variable, followed by one or two integers in parentheses, separated by a comma, indicating the place of that variable in the list. Up to 26 arrays are possible in any program (corresponding to the letters of the alphabet), subject only to the amount of memory space available for data storage. For example, a list might be described as A(I) where I goes from 1 to 5, as follows:

A(1),A(2),A(3),A(4),A(5)

This allows the programmer to reference each of the five elements in the list A. A two dimensional matrix A(I,J) can be defined in a similar manner, but the subscriped variable A can only be used once (i.e., A(I) and A(I,J) cannot be used in the same program). It is possible however, to use the same variable name as both a subscripted and an unsubscripted variable. That is, both A and A(I) are valid variable names for use in the same program.

Subscripted variables allow data to be input quickly and easily, as illustrated in the following program (the index of the FOR statement in lines 20, 42 and 44 is used as the subscript):

```
10 REM - PROGRAM DEMONSTRATING READING
11 REM - OF SUBSCRIPTED VARIABLES
15 DIM A(5),B(2,3)
18 PRINT "A(I) WHERE A=1 TO 5;"
20 FOR I=1 TO 5
25 READ A(I)
30 PRINT A(I);
35 NEXT I
38 PRINT
39 PRINT
40 PRINT "B(I,J) WHERE I=1 TO 2:"
41 PRINT "         AND  J=1 TO 3:"
42 FOR I=1 TO 2
```

```
43 PRINT
44 FOR J=1 TO 3
48 READ B(I,J)
50 PRINT B(I,J);
55 NEXT J
56 NEXT I
60 DATA 1,2,3,4,5,6,7,8
61 DATA 8,7,6,5,4,3,2,1
65 END
RUN
A(I) WHERE A=1 TO 5;
 1  2  3  4  5

B(I,J) WHERE I=1 TO 2:
         AND  J=1 TO 3:

 6  7  8
 8  7  6
READY.
```

DIM

From the preceding example, it can be seen that the use of subscripts requires a dimension (DIM) statement to define the maximum number of elements in the array. The DIM statement is of the form:

(line number) DIM v1(n1), v2(n2,m2)

where v indicates an array variable name and n and m are integer numbers indicating the largest subscript value required during the program. For example:

```
15 DIM A(6,10)
```

The first element of every array is automatically assumed to have a subscript of zero. Dimensioning A(6,10) sets up room for an array with 7 rows and 11 columns. This matrix can be thought of as existing in the following form:

```
A0,0 A0,1 ...  A0,10
A1,0 A1,1 ...  A1,10
A2,0 A2,1 ...  A2,10
   .              .
   .              .
   .              .
A6,0 A6,1 ...  A6,10
```

and is illustrated in the following program:

```
10 REM - MATRIX CHECK PROGRAM
15 DIM A(6,10)
20 FOR I=0 TO 6
22 LET A(I,0)=I
25 FOR J=0 TO 10
28 LET A(0,J)=J
30 PRINT A(I,J);
35 NEXT J
40 PRINT
45 NEXT I
50 END

RUN
 0  1  2  3  4  5  6  7  8  9  10
 1  0  0  0  0  0  0  0  0  0  0
 2  0  0  0  0  0  0  0  0  0  0
 3  0  0  0  0  0  0  0  0  0  0
 4  0  0  0  0  0  0  0  0  0  0
 5  0  0  0  0  0  0  0  0  0  0
 6  0  0  0  0  0  0  0  0  0  0

READY.
```

Notice that a variable assumes a value of zero until another value has been assigned. If the user wishes to conserve memory space by not making use of the extra variables set up within the array, he should set his DIM statement to one less than necessary, i.e. DIM A(5,9). This results in a 6 by 10 array which may then be referenced beginning with the A (0,0) element.

More than one array can be defined in a single DIM statement:

```
10 DIM A(20), B(4,7)
```

This dimensions both the list A and the matrix B.

A number must be used to define the maximum size of the array. A variable inside the parentheses is not acceptable and will result in an error message by BASIC at run-time. The amount of memory not filled by the program will determine the amount of data the computer can accept as input to the program at any one time. In some programs a TOO-BIG ERROR may occur, indicating that memory will not hold an array of the size requested. In that event, the user should change his program to process part of the data in one run and then chain to another section to process the rest (see Section 6.8.10).

6.8.9 Transfer of Control Statements

Certain control statements cause the execution of a program to jump to a different line either unconditionally or as a result of some condition within the program. Looping is one method of jumping to a designated point until a condition is met. The following statements give the programmer added capabilities in this area.

Unconditional Transfer (GOTO)

The GOTO (or GO TO) statement is an unconditional statement used to direct program control either forward or back in a program. The form of the GOTO statement is:

(line number) GOTO n

where n represents a statement number. When the logic of the program reaches the GOTO statement, the statement(s) immediately following will not be executed; instead execution is transferred to the statement beginning with the indicated line number.

The following program never ends; it does a READ, prints something, and jumps back to the READ via a GOTO statement. It atempts to do this over and over until it runs out of data, which is an acceptable, though not advisable, way to end a program.

```
10 REM - PROGRAM ENDING WITH ERROR
11 REM - MESSAGE WHEN OUT OF DATA
20 READ X
25 PRINT "X="X,"X↑2="X↑2
30 GO TO 20
35 DATA 1,5,10,15,20,25
40 END

RUN
X= 1           X↑2= 1
X= 5           X↑2= 25
X= 10          X↑2= 100
X= 15          X↑2= 225
X= 20          X↑2= 400
X= 25          X↑2= 625

DATA ERROR AT LINE 20

READY.
```

Conditional Transfer (IF THEN and IF GOTO)

A program sometimes requires that two values be compared at some point; control of program execution may be directed to different procedures depending upon the result of the comparison. In computing, values are logically tested to see whether they are equal, greater than, less than another value, or possibly a combination of the three. This is accomplished by use of the relational operators discussed in Section 6.5.3.

IF THEN and IF GOTO statements allow the programmer to test the relationship between two formulas (variables, numbers, or expressions). Providing the relationship described in the IF statement is true at the point it is tested, control will be transfered to the line number specified, or the indicated operation will be performed. The statements are of the form:

(line number) IF v1 <relation> v2 GOTO [or THEN] x

where v1 and v2 represent variable names or expressions, and x represents a line number or an operation to be performed. The use of either THEN or GOTO is acceptable.

The following two examples are equivalent (the value of the variable A is changed or remains the same depending upon A's relation to B):

```
  .
  .
100 IF A>B THEN 120
110 A=A↑B-1
120 C=A/D
  .
  .
```

```
  .
  .
100 IF A<=B THEN A=A↑B-1
110 C=A/D
  .
  .
```

IF END#

The IF END# statement is used to verify an end-of-file condition during run-time input. The form of this statement is:

(line number) IF END# THEN n

IF END# instructs BASIC to perform a check on the validity of the last INPUT# statement referencing the currently open input file; n represents a line number or operation to be performed. If an end-of-file (CTRL/Z) was detected during the last INPUT# statement, BASIC transfers control to the specified line number or performs the indicated operation. If an end-of-file was not detected, then no operation occurs. For example:

```
  .
  .
150 OPEN "1:VALUE" FOR INPUT
  .
  .
200 INPUT# A,B,C
210 IF END# THEN 530
215 LET X=SGN(A)
  .
  .
530 PRINT "INPUT FILE--NOT ENOUGH DATA"
535 STOP
  .
  .
```

In this example the programmer provides his own error message if there is an insufficient number of values for his variables. If there are two valid numbers remaining in the input file when statement 200 is reached, then the variables A and B will receive valid input. When the program attempts to input a value for C, BASIC will detect an end-of-file and return a value of zero for C. As it executes the IF END# statement, BASIC will note that it has just reached the end-of-file, and will transfer control to statement number 530, as the user intended.

However, assume that as line 200 is executed there is only one valid data value left in the input file. An end-of-file is detected this time when BASIC tries to read a value for B; B is set to zero. When BASIC attempts to continue reading a value for C, an EOF ERROR will be returned (see Section 6.12) and program execution will terminate since the user has tried to read past the end-of-file. A good way of circumventing this condition is to include both the INPUT# and the IF END# statements in a loop and input one value at a time. Using this method allows the programmer's own error message to be printed before BASIC is allowed to read past the end-of-file.

6.8.10 Program Chaining (CHAIN)

Since Cassette BASIC allows at most only 2K words of memory for program storage, it is possible that a program may be too large to fit in memory at one time. However, Cassette BASIC compensates for this by allowing different segments of a program to be stored on cassette and called as needed. Although each program segment is restricted to 2K of memory, total program length is effectively unlimited. The form of the CHAIN statement is:

(line number) CHAIN "n:XXXX"

where n is the cassette drive number, and XXXX is the name of the file to be chained to. The CHAIN statement should be the last statement in the user's program. When BASIC transfers to the program specified in the statement, it removes the old program from memory. Data is not passed in memory during the chain, so the user should be careful to save any data he will need in an output file. (See Section 6.8.5--PRINT#.) The chain automatically closes any open output file, transfers control to the lowest statement number in the new program, and continues execution.

For example, the following section of a program stores some data values on an output cassette and chains to a file called PART2:

```
    .
    .
    .
450 OPEN "1:DATA" FOR OUTPUT
455 COMMAS
460 PRINT# B,C,D,G,H,Z
465 NO COMMAS
470 FOR I=1 TO 10
475 PRINT# A(I)
480 NEXT I
485 CLOSE
490 CHAIN "1:PART2"
```

The values stored by this section of the program in the cassette file DATA can be read in by the second section of the program--PART2--and can continue to be used. PART2 might appear as follows:

```
1 DIM A(10)
5 OPEN "1:DATA" FOR INPUT
10 INPUT# B,C,D,G,H,Z
15 FOR I=1 TO 10
20 INPUT# A(I)
25 NEXT I
  .
  .
  .
```

6.8.11 Subroutines (GOSUB and RETURN)

A subroutine is a section of code performing some operation that is required at more than one point in the program. Often a complicated I/O operation for a volume of data, a mathematical evaluation which is too complex for a user-defined function, or any number of other processes may best be performed in a subroutine.

Subroutines are generally placed physically at the end of a program, usually before DATA statements, if any, and always before the END statement. Two statements are used exclusively in BASIC to handle subroutines; these are the GOSUB and RETURN statements.

A program begins execution and continues until it encounters a GOSUB statement of the form:

(line number) GOSUB x

where x represents the first line number of the subroutine. Control then transfers to that line. For example:

```
50 GOSUB 200
```

When program execution reaches line 50, control transfers to line 200, and the subroutine is processed until execution encounters a RETURN statement of the form:

(line number) RETURN

The RETURN statement causes control to return to the statement following the GOSUB statement. Before transferring to the subroutine, BASIC internally records the next statement to be processed after the GOSUB statement; thus the RETURN statement is a signal to transfer control to this statement. In this way, no matter how many different subroutines are called, or how many times they are used, BASIC always knows where to go next.

The following program demonstrates a simple subroutine:

```
1 REM - THIS PROGRAM ILLUSTRATES GOSUB AND RETURN
10 DEF FNA(X)=ABS(INT(X))
20 INPUT A,B,C
30 GOSUB 100
40 LET A=FNA(A)
50 LET B=FNA(B)
60 LET C=FNA(C)
70 PRINT
80 GOSUB 100
90 STOP
100 REM - THIS SUBROUTINE PRINTS OUT THE SOLUTIONS
110 REM - OF THE EQUATION: A(X↑2) + B(X) + C = 0
120 PRINT "THE EQUATION IS    "A"*X↑2   +   "B"*X   +  "C
130 LET D=B*B-4*A*C
140 IF D<>0 THEN 170
150 PRINT "ONLY ONE SOLUTION... X ="-B/(2*A)
160 RETURN
170 IF D<0 THEN 200
180 PRINT "TWO SOLUTIONS... X =";
185 PRINT (-B+SQR(D))/(2*A)"AND X ="(-B-SQR(D))/(2*A)
190 RETURN
200 PRINT "IMAGINARY SOLUTIONS... X = (";
205 PRINT -B/(2*A)","SQR(-D)/(2*A)") AND (";
207 PRINT -B/(2*A)","-SQR(-D)/(2*A)")"
210 RETURN
900 END

RUN
?1,.5,-.5
THE EQUATION IS     1 *X↑2   +    .5 *X   +  -.5
TWO SOLUTIONS... X = .5 AND X =-1

THE EQUATION IS     1 *X↑2   +    0 *X   +   1
IMAGINARY SOLUTIONS... X = ( 0 , 1 ) AND ( 0 ,-1 )

READY.
```

Line 100 begins the subroutine. There are several places in which control may return to the main program, depending upon the flow of control through the various IF statements. The subroutine is called from line 30 and again from line 80. When control returns to line 90, the program encounters the STOP statement and execution is terminated. It is important to remember that subroutines should generally be kept distinct from the main program. The last statement in the main program should be a STOP or GOTO statement, and subroutines are normally placed following this statement. A useful practice is to assign distinctive line numbers to subroutines. For example, if the main program is numbered with line numbers up to 199, then 200 and 300 could be used as the first numbers of two subroutines.

Nesting Subroutines

Nesting of subroutines occurs when one subroutine calls another subroutine. If a RETURN statement is encountered during execution of a subroutine, control returns to the statement following the GOSUB

which called it. From this point, it is possible to transfer to the beginning or any part of a subroutine, even back to the calling subroutine. Multiple entry points and RETURN statements make subroutines more versatile.

The maximum level of GOSUB nesting is about ten (decimal) levels, which should prove more than adequate for all normal uses. Exceeding this limit will result in the message:

GOSUB ERROR AT LINE XXXX

where XXXX represents the line number where the error occurred. An example of GOSUB nesting follows. Execution has been stopped by typing a CTRL/SHIFT/P combination (see Section 6.11.4, Stopping a Run), as the program would otherwise continue in an infinite loop.

```
10 REM - FACTORIAL PROGRAM USING GOSUB TO
15 REM - RECURSIVELY COMPUTE THE FACTORS
40 INPUT N
50 IF N>20 THEN 120
60 X=1
70 K=1
80 GOSUB 200
90 PRINT "FACTORIAL "N" ="X
110 GO TO 40
120 PRINT "MUST BE 10 OR LESS"
130 GO TO 40
200 X=X*K
210 K=K+1
220 IF K<=N THEN GOSUB 200
230 RETURN
240 END

RUN
?2
FACTORIAL  2   = 2
?4
FACTORIAL  4   = 24
?5
FACTORIAL  5   = 120
?

STOP.
READY.
```

6.8.12 Functions

BASIC defines several mathematical calculations for the programmer, eliminating the need for tables of trig functions, square roots, and logarithms. These functions have a 3-letter call name, followed by an argument, x, which can be a number, variable, expression, or another function. Table 6-1 lists the functions available in Cassette BASIC. Most are self-explanatory; those that are not and are described in greater detail are marked with an asterisk.

Table 6-1 Cassette BASIC Functions

Function	Meaning
SIN(x)	Sine of x (x is expressed in radians)
COS(x)	Cosine of x (x is expressed in radians)
TAN(x)	Tangent of x (x is expressed in radians)
ATN(x)	Arctangent of x (result is expressed in radians)
EXP(x)	e to the xth power (e=2.718282)
LOG(x)	Natural log of x ($\log_e x$)
*SGN(x)	Sign of x--assign a value of +1 if x is positive, 0 if x is zero, or -1 if x is negative
*INT(x)	Integer value of x
ABS(x)	Absolute value of x ($\|x\|$)
SQR(x)	Square root of x ($\sqrt{x}$)
*RND(x)	Random number
*TAB(x)	Print next character at space x
*GET(x)	Get a character from input device
*PUT(x)	Put a character on output device
*FNA(x)	User-defined function
*UUF(x)	User-coded function (machine language code)

Sign Function (SGN(x))

The sign function returns the value +1 if x is a positive value, 0 if x is zero, and -1 if x is negative. For example, SGN(3.42)=1, SGN(-42)=-1, and SGN(23-23)=0. The following example illustrates the use of this function:

```
10 REM - SGN FUNCTION EXAMPLE
20 READ A,B
25 PRINT "A="A,"B="B
30 PRINT "SGN(A)="SGN(A),"SGN(B)="SGN(B)
40 PRINT "SGN(INT(A))="SGN(INT(A))
50 DATA -7.32, .44
60 END
```

Integer Function (INT(x))

The integer function returns the value of the nearest integer not greater than x. For example, INT(34.67)=34. By specifying INT(x+.5) the INT function can be used to round numbers to the nearest integer; thus, INT(34.67+.5)=35. INT can also be used to round numbers to any given decimal place by specifying:

```
INT(X*10↑D+.5)/10↑D
```

where D is the number of decimal places desired. The following program illustrates this function; execution has been stopped by typing a CTRL/SHIFT/P:

```
10 REM - INT FUNCTION EXAMPLE
20 PRINT "NUMBER TO BE ROUNDED";
30 INPUT A
40 PRINT "NO. OF DECIMAL PLACES";
50 INPUT D
60 LET B=INT(A*10↑D+.5)/10↑D
70 PRINT "A ROUNDED = "B
80 GO TO 20
90 END

RUN
NUMBER TO BE ROUNDED?55.65342
NO. OF DECIMAL PLACES?2
A ROUNDED =  55.65
NUMBER TO BE ROUNDED?78.375
NO. OF DECIMAL PLACES?-2
A ROUNDED =  100
NUMBER TO BE ROUNDED?67.89
NO. OF DECIMAL PLACES?-1
A ROUNDED =  70
NUMBER TO BE ROUNDED?
STOP.
READY.
```

If the argument is a negative number, the value returned is the largest negative integer (rounded to the higher value) contained in the number. For example, INT(-23)=-23 but INT(-14.39)=-15.

Random Number Function (RND(x))

The random number function produces a random number n which is in the range 0<n<1. The numbers are not reproducible, a fact the programmer should keep in mind when debugging or checking his program. The argument x in the RND(x) function call can be any number, as that value is ignored. The following program illustrates the use of this function to generate a table of random numbers.

```
10 REM - RANDOM NUMBER EXAMPLE
25 PRINT "RANDOM NUMBERS"
30 FOR I=1 TO 30
40 PRINT RND(0),
50 NEXT I
60 END

RUN
RANDOM NUMBERS
 .7759228       .08069808      .5008833       .2790171       .1661529
 .4857633       .4192038       .1433537       .08728769      .2335427
 .6156673       .5921191       .01170888      .7411813       .341708
 .3796163       .2023254       .7974058       .9635064       .6043865
 .9547609       .2890875       .1416765       .2482717       .2145417
 .05280478      .3859534       .8404774       .5692836       .8514056
READY.
```

It is possible to generate random numbers over any range by using the following formula:

```
(B-A)*RND(0)+A
```

This produces a random number (n) in the range A<n<B. For example, in order to obtain random digits in the range 0<n<9, line 40 in the previous example is changed to read:

```
40 PRINT 9*RND(0),
```

To obtain random integer digits, the INT function is used in conjunction with the RND function (using the same values for A and B above) as follows:

```
40 PRINT INT(9*RND(0)),
```

When the program is run, the results will look as follows:

```
RUN
RANDOM NUMBERS
 4            5            8            1            8
 1            0            7            8            5
 8            3            0            8            8
 2            4            7            7            4
 8            7            1            8            2
 0            7            0            7            5
READY.
```

Notice that the range has changed to 0<=n<9. This is because the INT function returns the value of the nearest integer not greater than n.

Tab Function (TAB(n))

The TAB function allows the user to position the printing of characters anywhere on the teleprinter (or line printer) line. Print positions can be thought of as being numbered from 1 to 72 across the console terminal line (1 to 80 across the line printer line) from left to right. The form of this function is:

```
TAB(n)
```

where the argument n represents the position (from 1 to the total number of spaces available) in which the next character will be typed. For example, TAB(3) causes the character to be printed at position 3.

Each time the TAB function is used, positions are counted from the beginning of the line, not from the current position of the printing head. For example, the following statement:

```
10 PRINT "X =";TAB(3);"/";3.14159
```

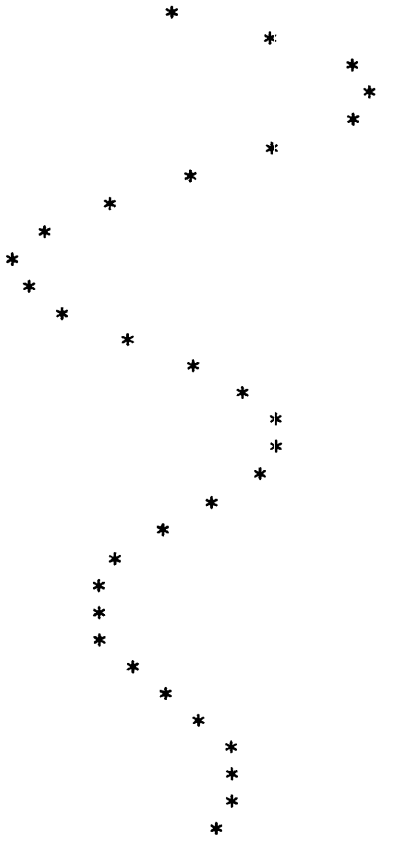
will print the slash on top of the equal sign, as shown below:

```
RUN
X = 3.14159
```

The following is an example of the sort of graph that can be drawn with BASIC using the TAB function:

```
30 FOR X=0 TO 15 STEP .5
40 PRINT TAB(30+15*SIN(X)*EXP(-.1*X));"*"
50 NEXT X
```

```
RUN
```



PUT and GET Functions (PUT(x), GET(x))

Cassette BASIC provides two additional functions, PUT and GET, to increase input/output flexibility on the console terminal or line printer. Using these statements, the programmer can "PUT" an ASCII character on the current output device, or "GET" a character from the current input device. (ASCII character codes are listed in Appendix A.) GET is of the form:

GET(x)

where the argument x is a dummy variable which may be any value. GET(x) will be assigned the decimal value of the ASCII code of the next character input on the current input device.

For example, if the following statement appears in a program:

```
10 LET L=GET(0)
```

and the next character input is an M, the variable L will be assigned the value 77(decimal).

PUT is of the form:

PUT(x)

where the argument x represents either the decimal value of the ASCII code of the character to be output, or the character itself. For example, the statement:

```
15 L=PUT(GET(V))
```

will wait for a character to be read from the current input device and then print it on the current output device. A statement such as:

```
30 PRINT PUT(Q)
```

will print the character typed as well as the decimal value of the ASCII code for that character. (Since both the character and the decimal value are typed, PUT and GET statements should not be used with cassette files.)

NOTE

If the user is inputting characters from paper tape via the paper tape reader on an LT33 Teletype, he should be careful to position the tape on the first character to be input. Otherwise blank tape may be entered, which is interpreted as a "BREAK" and stops a running program.

The PUT function can also be used to format output. For example, to print a trig table on the line printer with a heading and 50 data lines per page, the form feed character (12 decimal) can be "PUT" to the printer as follows:

```
100 LPT
110 GOSUB 1000
120 GOSUB 500
125 REM - SET UP TRIG TABLE
130 FOR J=0 TO 360 STEP .5
140 LET L=L+1
150 LET B=J/180*3.14
160 PRINT J,SIN(B),COS(B),TAN(B),ATN(B)
165 REM - PRINT 50 ENTRIES IN TABLE
170 IF L=50 THEN GOSUB 500
180 NEXT J
190 GOSUB 1000
200 GOSUB 1000
210 STOP
500 REM - PRINT HEADER
505 GOSUB 1000
510 PRINT
520 PRINT
530 PRINT "ANGLE","SINE","COSINE","TANGENT","ARCTANGENT"
540 PRINT
550 RETURN
1000 REM - PRINT FORM FEEDS TO ADVANCE PAPER
1005 X=PUT(12)
1010 L=0
1020 RETURN
1030 END
```

The beginning of the line printer output from this program follows. The first page of the table continues through an angle of 24.5 degrees: then the header and the next 50 entries are printed on the next page, and so on until the values have been output (in steps of .5) for all angles through 360 degrees.

ANGLE	SINE	COSINE	TANGENT	ARCTANGENT
0	0	1	0	0
.5	8.722112E-03	.999962	8.722444E-03	8.722001E-03
1	.01744356	.9998479	.01744621	.01744268
1.5	.02616368	.9996577	.02617264	.0261607
2	.03488181	.9993915	.03490305	.03487474
2.5	.04359729	.9990492	.04363878	.0435835
3	.05230945	.9986309	.05238116	.05228564
3.5	.06101763	.9981367	.06113154	.06097986
4	.06972117	.9975665	.06989125	.06966486
4.5	.0784194	.9969205	.07866164	.07833935
5	.08711167	.9961986	.08744408	.08700204
5.5	.09579731	.9954009	.09623993	.09565166
6	.1044757	.9945274	.1050506	.1042869
6.5	.1131461	.9935784	.1138774	.1129067
7	.1218079	.9925537	.1227217	.1215095
7.5	.1304604	.9914535	.131585	.1300944
8	.139103	.9902779	.1404687	.13866
8.5	.147735	.989027	.1493741	.1472052
.				
.				
.				
24.5	.414496	.9100512	.4554645	.4038923

FNA Function (DEF FNA(x))

In some programs it may be necessary to execute the same mathematical formula in several different places. Cassette BASIC allows the programmer to define his own function in the BASIC language and then call this function in the same manner as the square root or a trig function is called. Only one such user-defined function may be included per program. The function is defined once at the beginning of the program before its first use, and consists of a DEF statement in combination with a 3-letter function name, the first two letters of which must be FN. The format of the DEF FNA statement is as follows:

(line number) DEF FNA(x)=formula(x)

The A in the FNA portion of the statement may be any letter. The argument (x) has no significance; it is strictly a dummy variable but must be the same on each side of the equal sign. The function itself can be defined in terms of numbers, several variables, other functions, or mathematical expressions. For example:

```
10 DEF FNA(X)=X↑2+3*X+4
```

or

```
20 DEF FNC(X)=SQR(X+4)+1
```

The function:

```
10 DEF FNL(S)=S↑2
```

will cause the later statement:

```
20 LET R=FNA(4)+1
```

to be evaluated as R=17.

The user-defined function can be a function of only one variable.

User-Coded Function (UUF)

The user-coded function is explained in detail in the next section.

6.9 IMPLEMENTING A USER-CODED FUNCTION (UUF)

A special user-coded function is available in Cassette BASIC for the programmer who is familiar with the PDP-8 instruction set and 27-bit mantissa floating-point format. BASIC's internal format is 27-bit, sign-magnitude mantissa floating-point; thus, all user-generated values must be in that format and all coding must be compatible with it. The user codes the function in the PDP-8 series machine language instructions, assembles it with the PALC Assembler, and loads the resulting binary file as an overlay to one of the existing functions (ATAN, LOG, etc.) Thus, while BASIC is running, this special function can be requested and used in a fashion analogous to the built-in BASIC functions. The user-coded function, if present, is specified in the BASIC program as:

UUF(n)

where n can be any BASIC expression.

6.9.1 Coding Formats

Due to memory restrictions, the user-coded function must replace one or more of the existing Cassette BASIC functions. Table 6-2 lists the functions which may be overlaid and the areas of memory they occupy. Also listed is the transfer table address through which BASIC calls the given extended function.

Table 6-2 Function Addresses

Function	Locations Occupied	Transfer Table Address
FNA	5453-5546	1131
ATN	6200-6271	1134
SQR	5412-5452	1137
RND	5350-5406	1143
TAB	5547-5572	1147

The functions SIN, COS, and TAN are interdependent, but all three may be deleted as follows:

SIN	5600-5674	1132
COS		1133
TAN		1144

Almost a full page is freed by deleting the following:

FNA	5412-5572	1131
SQR		1137
TAB		1147

For each function replaced by the UUF, the user must set the corresponding transfer table location to point to an error routine so that accidental calls to that function will generate an error condition rather than a spurious call to the UUF. The user does this by inserting a statement such as the following in his UUF:

```
*1143       /TABLE ADDRESS FOR RND
6441        /POINTER TO SYNTAX ERROR ROUTINE
```

To include a user-coded function, all conventions required for the PALC Assembler must be observed. The coding language is PDP-8 machine language code, but can include instructions in the modified floating-point package, which is described later in this chapter in Section 6.10.

When floating-point statements are to be included in the program, it is necessary only to indicate the start of floating-point notation by including the following operator:

FENTER

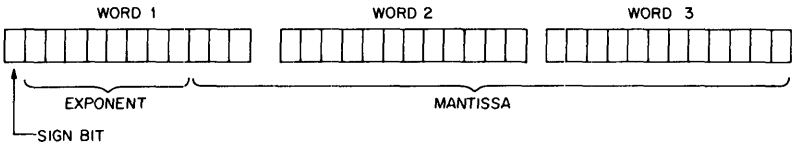
immediately before the first floating-point statement. Similarly floating-point coding is terminated by the operator:

FEXIT

immediately after the last floating-point statement. There can be as many sections of floating-point code as necessary in the program, but each must be delimited in this manner.

6.9.2 Floating-Point Format

The floating-point format used by Cassette BASIC allocates three storage words in sign magnitude convention as follows (in sign magnitude convention the sign bit rather than the mantissa, expresses the sign of the entire number):



Five memory locations are used to represent the floating-point accumulator, as follows:

Table 6-3 Floating-Point Accumulator

Location Name	Value	Contents
ACS	0024	Sign
ACE	0025	Exponent (200 octal biased; i.e. the constant 200 is added to the exponent to make the range 0-377)
AC1	0020	High order word
AC2	0017	Mid order word
AC3	0016	Low order word

All of BASIC's mathematical operations are in floating-point format; therefore, if any temporary storage locations are required by the UUF subroutine, they must specify three words. For example:

```
UTEMP,0;0;0
```

6.9.3 Incorporating Subroutines with UUF

When adding a user function, it becomes necessary to reference some of Cassette BASIC's subroutines at specific times in the coding. Most of these calls are needed in order to preserve a compatible format throughout the system. The BASIC subroutines which may be referenced are described below. (The complete BASIC symbol table is included as Table 6-7 at the end of this chapter.)

BEGFIX

If a value is to be returned to the accumulator as a result of the user function, that value must be in normalized floating-point format. If floating-point arithmetic is used throughout the user function, then the value in the FAC (floating-point accumulator) is in normalized floating-point format and need not be converted. If fixed-point arithmetic is used anywhere in the function, then the subroutine BEGFIX must be referenced before the value (floating-point) is saved in order that the storage locations are properly initialized to accept a floating-point value. Using this procedure, the five FAC locations are prepared accordingly. However, because the value to be stored only requires 12 bits, a subsequent DCA AC3 statement is sufficient. BEGFIX is located at 3760 and is called via a JMS instruction.

ANORM

If a fixed-point value is added to the FAC, ANORM normalizes the FAC in order that it be in a format suitable for Cassette BASIC. The routine supplies the acceptable values for the locations ACE, ACS, AC1 and AC2. ANORM is assigned the location 4600.

FIX

To convert a value in the FAC to an integer, as when printing a character, the subroutine FIX is called; it is located at 4744.

6.9.4 Writing the Program

A user-coded function must reset one of Cassette BASIC's tables to recognize the function, otherwise, UUF is considered to be an undefined function. The pointer is at location 1150; a statement such as the following is required:

```
*1150
UUF
```

Procedures for loading a user-coded function are contained in Section 6.11.5. Examples of user-coded functions follow.

6.9.5 Examples of User-Coded Functions

Example 1--This program calculates squares and square roots for a series of values. The BASIC program is as follows:

```
100 FOR A=33.1 TO 33.9 STEP .1
110 PRINT A,UUF(A),SQR(A)
120 NEXT A
130 END
```

The user-coded function is:

```
                PALC-V1  12/27/72  PAGE 1

                /
                /USER-CODED FUNCTION TO CALCULATE
                /SQUARES OF NUMBERS
                /
                /THE FUNCTION LOADS INTO FIELD 0
                /INTO THE AREA OCCUPIED BY THE 'ATN'
                /FUNCTION
        4435    FENTER=4435
        2000    FST=2000
        0200    FWD=200
        6000    FMP=6000
        0000    FEXIT=0000
        6441    SXERR=6441
        1134            *1134
01134   6441            SXERR   /SO REFERENCES TO ATN WILL
                                /YIELD AN ERROR
        1150            *1150
01150   6200            UUF     /DEFINE UUF IN FUNCTION TABLE

        6200            *6200
06200   0000    UUF,    0
06201   4435            FENTER  /INTO FLT.PT.PKG.-A IS IN FAC
06202   2204            FST+FWD+X-.     /SAVE A
06203   6203            FMP+FWD+X-.     /A*A
06204   0000            FEXIT
06205   5600            JMP I   UUF     /ALL DONE
06206   0000    X,      0;0;0
06207   0000
06210   0000
        0001    FIELD 1
        3000    *3000   /TO START BASIC WHEN LOADED
                $

/                                       PALC-V1  12/27/72  PAGE 1-1

FENTER  4435
FEXIT   0000
FMP     6000
FST     2000
FWD     0200
SXERR   6441
UUF     6200
X       6206
```

The output after execution is:

```
RUN
 33.1           1095.61         5.75326
 33.2           1102.24         5.761944
 33.3           1108.89         5.770615
 33.4           1115.56         5.779273
 33.5           1122.25         5.787918
 33.6           1128.96         5.796551
 33.7           1135.69         5.80517
 33.8           1142.44         5.813777
 33.9           1149.21         5.822371

READY.
```

Example 2--The following program tests a student's ability to convert the octal value in the console switches to its decimal equivalent. Line 120 will set P equal to the decimal value of the setting. Line 130 determines if the correct answer was typed:

```
100 PRINT "WHAT DECIMAL VALUE DO THE SWITCHES EQUAL?"
110 INPUT N
120 LET P=UUF(0)
130 IF P=N THEN 200
140 PRINT "TRY AGAIN"
150 GO TO 100
200 PRINT "CORRECT"
300 END
```

The user-coded function is:

```
                                        PALC-V1  12/27/72  PAGE 1

                    /
                    /USER-CODED FUNCTION TO READ THE CONSOLE
                    /SWITCHES AND CONVERT TO FLOATING POINT
                    /
                    /THE FUNCTION LOADS INTO FIELD ZERO
                    /INTO THE AREA PREVIOUSLY OCCUPIED BY
                    /THE 'RND' FUNCTION-THE RANDOM NUMBER
                    /GENERATOR
                    /
             0000   FIELD 0
             4435   FENTER=4435
             0000   FEXIT=0000
             4000   FAD=4000
             2000   FST=2000
             0200   FWD=200
             3760   BEGFIX=3760
             4600   ANORM=4600
             0016   AC3=16
             6441   SXERR=6441

             1143           *1143
01143        6441           SXERR    /SO REFERENCES TO RND
                                     /WILL YIELD AN ERROR
             1150           *1150
01150        5350           UUF      /DEFINE UUF IN FUNCTION TABLE

             5350           *5350
05350        0000   UUF,    0
05351        4756           JMS I    UBEGFIX /PREPARE FOR
                                             /INTEGER VALUE
05352        7604           LAS              /GET CONTENTS
                                             /OF SW. REG.
05353        3016           DCA      AC3     /SAVE IN LOW
                                             /ORDER FAC
05354        4757           JMS I    UANORM  /NORMALIZE
05355        5750           JMP I    UUF     /RETURN
05356        3760   UBEGFIX,BEGFIX
05357        4600   UANORM, ANORM
             0001   FIELD 1
             3000   *3000    /TO START UP BASIC WHEN LOADED
                    $

                                        PALC-V1  12/27/72  PAGE 1-1

AC3     0016
ANORM   4600
BEGFIX  3760
FAD     4000
FENTER  4435
FEXIT   0000
FST     2000
FWD     0200
SXERR   6441
UANORM  5357
UBEGFI  5356
UUF     5350
```

An example of a run in which 200(octal) was set in the console switches follows:

```
RUN
WHAT DECIMAL VALUE DO THE SWITCHES EQUAL?
?120
TRY AGAIN
WHAT DECIMAL VALUE DO THE SWITCHES EQUAL?
?128
CORRECT

READY.
```

6.10 FLOATING-POINT PACKAGE

Information concerning the PDP-8 modified Floating-Point Package which the programmer will find useful in coding a function follows.

6.10.1 Instruction Set

The legal instructions in the modified Floating-Point Package used by Cassette BASIC are explained in Table 6-4:

Table 6-4 Floating-Point Instructions

Instruction	Value	Meaning
FST	2000	Store the contents of the floating accumulator (FAC). The contents of the FAC are not changed.
FLD	3000	Load FAC with contents of relative address.
FAD	4000	Add contents of relative address to FAC.
FSB	5000	Subtract contents of relative address from FAC.
FMP	6000	Multiply the contents of the FAC by the contents of the relative address.
FDV	7000	Divide FAC by contents of relative address.
FJMP	1000	Floating-point jump to relative address.
FENTER	4435	Start floating-point code.
FEXIT	0000	Exit floating-point code. Return to PDP-8 code.
FWD	0200	Access a relative location in the forward direction.
BKWD	0600	Access a relative location in the backward direction.
FSNE	0040	Skip if FAC <> 0
FSEQ	0050	Skip if FAC = 0
FSGE	0100	Skip if FAC => 0
FSLT	0110	Skip if FAC <0
FSGT	0140	Skip if FAC >0
FSLE	0150	Skip if FAC <= 0

The following instructions require indirect (and relative) addressing and therefore only address field 1. Their operation is the same as the corresponding direct instruction.

Table 6-5 Relative Addresses

Instruction	Value	Operation
FSTI	2400	Store
FLDI	3400	Load
FADI	4400	Add
FSBI	5400	Subtract
FMPI	6400	Multiply
FDVI	7400	Divide
FJMPI	1400	Jump

6.10.2 Addressing

The Floating-Point Package uses relative addressing. Thus, all statements that address a location must include one of the operators FWD or BKWD plus a reference to the current location. Such a reference is generally in the form:

op code instruction + FWD (BKWD) + LTEMP-.

The operator FWD is used when the address of the location to be referenced is numerically greater than the current address; BKWD is used when the address of the location to be referenced is numerically less than the current address. LTEMP-. in conjunction with the FWD or BKWD operator defines the relative address of the location to be operated on (LTEMP) with respect to the current location. This relative displacement is then used by the Floating-Point Interpreter to access the contents of the three words at LTEMP. This can best be seen in an example:

```
4010 FAD+FWD+LTEMP-.
  .
  .
4063 LTEMP,0;0;0
```

The contents of that location which is (4063-4010) locations forward from the current address, (i.e. the contents of LTEMP), are added to the FAC. Similarly:

```
146 ALOC,0;0;0
  .
  .
152 FMP+BKWD+.-ALOC
```

At line 152 the contents of FAC are multiplied by the contents of the location that is (152-146) locations backward from the current address.

6.11 EDITING AND CONTROL COMMANDS

Errors made while typing at the console keyboard are easily corrected. BASIC provides special commands to facilitate the editing procedure.

6.11.1 Erasing Characters and Lines (SHIFT/O, RUBOUTS, NO RUBOUTS, ALTMODE)

There are two methods available for erasing a character or series of characters one at a time. Typing a SHIFT/O causes the deletion of the last character typed and echoes as a back arrow (←) on the LT33 (or 35) Teletype, or as an underscore (_) on most other console terminals. One character is deleted each time the key is typed.

The RUBOUT key (or DELETE key on some terminals) may also be used for deletion of characters one at a time providing the command:

```
RUBOUTS
```

has been typed on the keyboard before the editing is done. This command enables the RUBOUT key to be used and must be typed each time a new program is in memory. If the user has neglected to type this command, he may not use the RUBOUT key. A later command of:

```
NO RUBOUTS
```

disables the key for use.

For example:

```
10 LEB←T A=10*B
```

The user types a B instead of T and immediately notices the mistake. He may type SHIFT/O (or RUBOUT key, if enabled) once to delete the B, (and as many times more as characters, including spaces, are to be deleted). After the correction is made, he may continue typing the line. The typed line enters the computer only when the RETURN key is pressed. Before that time any number of corrections can be made to the line.

```
20 DEN F←←F FNA(X,Y)=X↑2+3*Y
```

When the RETURN key is typed, the line is input as:

```
20 DEF FNA(X,Y)=X↑2+3*Y
```

Notice that spaces, as well as printing characters, may be erased.

The user may erase an entire line (provided the RETURN key has not been typed) by typing the ALTMODE key (ESCAPE key on some keyboards). BASIC echos back:

```
DELETED
```

at the end of the line to indicate that the line has been removed. The user continues as though it were a new line. If the RETURN key has already been typed, the user may still correct the line by simply typing the line number and retyping the line correctly. He may delete the line by typing the RETURN key immediately after the line number, thus removing both the line number and line from his program.

If the line number of a line not needing correction is accidentlally typed, the SHIFT/O or RUBOUT key may be used to delete the number(s); the user may than type in the correct numbers. Assume the line:

```
10 IF A>5 GOTO 230
```

is correct. The programmer intends to insert a line 15, but instead types:

```
10 LET
```

He notices the mistake and makes the correction as follows:

```
10 LET←←←←←5 LET X=Z-3
```

Line 10 remains unchanged, and line 15 is entered.

Following an attempt to run a program, error messages may be output on the console terminal indicating illegal characters or formats, or other user errors in the program. Most errors can be corrected by typing the line number(s) and correction(s) and then rerunning the program. As many changes or corrections as desired may be made before runs.

6.11.2 Listing a Program (LIST, LIST and LPT)

An indirect program or data can be listed on the active output device by typing the command:

```
LIST
```

followed by the RETURN key. The entire program (or data) will be listed.

A part of a program may be listed by typing LIST followed by a line number. This causes that line and all following lines in the program to be listed. For example:

```
LIST 100
```

will list line 100 and all remaining lines in the program.

The LIST command may be used in conjunction with the LPT command as follows:

```
LPT
LIST
```

This will list the current program on the line printer. Control is reset to the console terminal after the listing is completed.

6.11.3 Running a Program (RUN, RUN and LPT)

After a BASIC program has been typed and is in memory, it is ready to be run. This is accomplished by simply typing the command:

```
RUN
```

followed by the RETURN key. The program will begin execution. If errors are encountered, appropriate error messages will be typed on the keyboard; otherwise, the program will run to completion, printing whatever output was requested. When the END statement is reached, BASIC stops execution and prints:

```
READY.
```

The line printer, if available, can be used in conjunction with the RUN command, as follows:

```
LPT
RUN
```

After this command is issued, all output during program execution is diverted from the console terminal to the line printer, eliminating the need of inserting the LPT statement within the program. The output device is reset to the console terminal after execution.

6.11.4 Stopping a Run (CTRL/C, CTRL/O, CTRL/SHIFT/P, BREAK)

To stop a program during execution or to return to the Keyboard Monitor at any time, type a CTRL/C (by pressing the CTRL key and the C key simultaneously). This causes the current operation to be aborted immediately and the Cassette Keyboard Monitor to be re-bootstrapped from the System Cassette.

The command CTRL/O (produced by typing the CTRL and O keys simultaneously) is used to stop teleprinter output temporarily. The program will continue to execute but output will not be printed unless an error occurs or unless BASIC is waiting for a command or for data from an input statement. In the latter case, the console terminal is the expected input device. This feature is particularly useful for programs that print lengthy introductions and then request a user-specified parameter. Typing CTRL/O after the program is started will cause BASIC to bypass printing the introduction and wait until

the parameter is specified, thereby saving the time required to print the message. A second CTRL/O will resume output.

NOTE

For most programs that do not wait for input from the terminal, processing of the program after an initial CTRL/O will be completed before a second CTRL/O can be typed. Thus, it is very possible for no output to be printed rather than the anticipated partial output.

Certain terminals (such as Teletype models LT33 and 35) are equipped with a BREAK key which may be used in Cassette BASIC to interrupt program execution. Pressing the BREAK key causes a halt in execution and a return to the BASIC Editor for more commands. For those systems containing terminals not equipped with the BREAK feature, the same result can be produced by pressing the CTRL, SHIFT, and P keys simultaneously.

6.11.5 Loading a User-Coded Function

A user-coded function is created using the CAPS-8 EDITOR; it is assembled using PALC. The resulting binary file is loaded with BASIC using the Monitor Run or Load commands as follows:

```
.R BASIC, drive #:filename
```

or

```
.L BASIC, drive #:filename
```

Assume a user-coded function called UUF.BIN is stored on cassette drive 3. Assume also that the file UUF.BIN has been coded so as to include the correct starting address for BASIC. The user runs BASIC loading the function as follows:

```
.R BASIC,UUF
```

The starting address for BASIC is included in the program and coded as follows:

```
FIELD 1
*3000
$
```

The new function may now be used in any files the user wishes to edit and run.

6.11.6 Erasing a Program in Memory (SCRATCH)

The command:

```
SCRATCH
```

or

```
SCR
```

is provided to allow the programmer to clear his storage area, deleting any commands or a program which may have been previously entered, and leaving a clean area in which to work. If the storage area is not cleared before entering a new program, lines from previous programs may be executed along with the new program, causing errors or misinformation. The SCRATCH command eliminates all old statements and numbers and should be used before any new programs are read into memory or created at the keyboard.

Note that the SCR command does not clear the program name. If the user wishes to create a new program with a new name, he should use the NEW command which also performs a SCRATCH.

6.11.7 Renaming a Program (NAME)

The user may change the name of the program in memory by issuing the command:

```
NAME
```

BASIC responds by asking:

```
NEW PROGRAM NAME-
```

The user specifies a new filename (and extension, if desired). This changes the name of the program without affecting its image in memory. All subsequent references to the program must use this new name.

6.11.8 Saving a Program (SAVE)

Once the user has created or edited a program, he may want to save the new version on a cassette for later use. He does this by typing:

```
SAVE
```

BASIC asks:

```
UNIT #(Ø-7):
```

to which the user responds with the number of the cassette drive on which he wishes the program to be stored. The program is saved under its current name--that is, the name used in BASIC's initial dialogue, or its new name if the NAME command has been used to change it. (If the filename is the same as one already present on the cassette, the old file is replaced by *EMPTY in the directory and the new file is

written onto the cassette.) After the program has been saved, it is still in memory and may be RUN or edited further.

Attempting to save a program on a full cassette causes BASIC to return to the editing phase; the user must save the program on another cassette.

If the user does not specify a name for his program in the initial dialogue (by responding with an ALT MODE to the NEW PROGRAM NAME request), the program will be saved under the assigned name NONAME.BAS.

If the user SCRATCHes a program, creates another program without assigning a name to it by use of the NEW or NAME commands, and then attempts to save it, it will be saved under the name of the last program which was in memory, possibly deleting that program if saved on the same drive.

6.12 CASSETTE BASIC ERROR MESSAGES

BASIC checks all commands before executing them. If for some reason it cannot execute a command, BASIC indicates this by typing one of the following error messages and the number of the line in which the error occured. The form is:

ERROR MESSAGE AT LINE XXXX

Table 6-6 lists the errors BASIC checks for and reports before execution.

Table 6-6 Cassette BASIC Error Messages

Message	Meaning
ARGUMENT ERROR	A function has been given an illegal argument; for example: SQR(-1)
CHAIN ERROR	A cassette error occurred while doing program chaining; the user should not attempt to run the program in memory again.
DATA ERROR	There were no more items in the data list.
EOF ERROR	An attempt was made to read past the end-of-file during run-time input. Program execution terminates and control returns to the Keyboard Monitor.
EXPRESSION ERROR	One of BASIC's internal lists overflowed while attempting to evaluate an expression.
FILE NAME ERROR	A mistake or illegal character was found in the user's specification of a cassette drive # or file name in either a CHAIN or an OPEN statement.

Table 6-6 Cassette BASIC Error Messages (Cont'd)

Message	Meaning
FILE OPEN ERROR	The user attempted to open a run-time output file when one was already open, or a hardware error occurred.
FOR ERROR	FOR loops were nested too deeply.
FUNCTION ERROR	The user attempted to call a function which had not been defined.
GOSUB ERROR	Subroutines were nested too deeply.
I O ERROR	The user attempted to do run-time input and output to the same cassette at the same time.
IN ERROR	A cassette error occurred while attempting to carry out an OLD command or while doing run-time input.
LINE TOO LONG	A line of more than 80 characters was entered; BASIC ignores the whole line and waits for the user to enter a new line.
LINE # ERROR	A GOTO, GOSUB, or IF referenced a nonexistent line.
LOOKUP ERROR	BASIC could not find a run-time input file on the drive specified.
NEXT ERROR	FOR and NEXT statements were not properly paired.
NO FILES ERROR	The user attempted to do run-time file I/O without first specifying so during BASIC's initial dialogue.
OUT ERROR	An error (probably end-of-tape) occurred while doing cassette output either during a SAVE or during run-time output. If the error occurred during a SAVE, the user should retry the SAVE to a different cassette. If the error occurred during run-time output, he should re-run his program using a different cassette for output.
RETURN ERROR	A RETURN statement was issued when not under control of a GOSUB.
SUBSCRIPT ERROR	A subscript has been used which is outside the bounds defined in the DIM statement.

Table 6-6 Cassette BASIC Error Messages (Cont'd)

Message	Meaning
SYNTAX ERROR	A command did not correspond to the language syntax. Common examples of syntax errors are misspelled commands, unmatched parentheses, and other typographical errors. Reference to an undefined UUF will also produce this diagnostic.
TOO BIG, LINE IGNORED	The combination of program size and number of variables exceeds the capacity of the computer. Reducing one or the other may help. Otherwise, the user must break his program into parts and chain them together. A large number of DATA statements might be put into a run-time input file.

The following programming errors are not reported by Cassette BASIC, but instead are used in the computation as specified. They are included here for the programmer's reference.

1. Attempting to use a number in a computation which is too large for BASIC to handle will produce a result which is meaningless.

2. Attempting to use a number in a computation which is too small for BASIC to handle will result in the value zero being used instead.

3. Attempting to divide by zero will produce a result which is meaningless.

6.13 CASSETTE BASIC SYMBOL TABLE

Table 6-7 lists the Cassette BASIC symbols and their values. This information is useful when writing user-coded functions.

Table 6-7 Cassette BASIC Symbol Table

/PDP-8/E CASSETTE BASIC PAL8-V8 12/27/72 PAGE 51

```
ABCDEF 1763     CLOSE  6400     OVLOOP 5245     FEXPF  6067
ABS    6425     CLOSEO 1170     OWRIT  6731     FEXPI  6061
ACE    0025     CMSWCH 0176     ECHO   7023     FEXPU  6064
ACN    4417     CNAMST 0452     EDIT   2412     FHER   5117
ACS    0024     CNCLR  0141     END    2400     FILALT 0367
AC1    0020     CNERR  0501     ENDLIN 5702     FILNAM 0600
AC2    0017     CNTFND 7060     ENDNM  7332     FILNR  1356
AC3    0016     CNTLO  7407     ENDNUM 3321     FIL1   0606
ADDRES 0071     CNTLOZ 0134     ENDPDL 5743     FINDIT 0560
ALGNLP 4466     CNTO   1301     ENTER  6404     FINDLU 0566
ALLOC  1460     COUELO 0004     EOFAD  4526     FIOER  7135
ALL3   3140     COLUMN 0130     EOFRTN 6650     FIX    4744
ALTMOD 2661     COMCK  2277     EPTR   0060     FIXEXI 4773
AL1    4554     COMMAS 6175     ERROR  4136     FIXITU 5200
AMATCH 6506     COMMON 3377     EVAL   1000     FIXLIN 2135
ANORM  4600     COMON  1123     EVALGO 1004     FIXLUP 4750
ARGERR 7247     CONST  1367     EXECUT 0213     FIXUP  5143
ARFLOC 0003     CONEIN 0423     EXIT   2407     FJMP   1000
AR1    4402     COS    5616     EXP    6000     FJMPI  1400
ASKAGN 1135     CONT   7143     EXPGOO 5242     FJUMP  1130
ATLINE 6451     CRINIX 3073     EXPLON 5764     FLD    3000
ATN    6200     CRLF   6531     EXPOK  5265     FLDI   3400
ATNHIG 6265     CRLFD  5257     EXTLOC 0224     FLOGC1 6172
ATNLOW 6220     CRLFPR 3736     FAD    4000     FLOGC2 6153
ATNNOT 6257     CTRLC  0005     FADEXT 1314     FLOGC3 6156
BADCHN 7064     CTRLCJ 7604     FADI   4400     FLOGC4 6161
BARROW 0764     CTRLZI 6652     FALT   0644     FMP    6000
BCKWDS 4502     CTRZCK 0113     FATNAX 6273     FMPI   6400
BEGFIX 3760     CTRZHP 0052     FATNC  6337     FMTENF 5121
BKPO   0600     CVTLOO 5022     FATNCH 6342     FMT1   5123
BREAK  6522     DATAER 1672     FATNCJ 6345     FMT2   5051
BSKIP  2735     DECEXP 0043     FATNC1 6304     FMT3   5126
BSW    7002     DECFRA 5343     FATNC2 6307     FN     5453
BUFEND 6200     DEEPER 0530     FATNC3 6312     FNERR  5172
BUFST  6000     DEF    1575     FATNC4 6315     FNEXIT 1200
CAM    7621     DELETE 6501     FATNC5 6320     FOR    0415
CARRET 2664     DEVCOM 7354     FATNC6 6323     FORCT  0065
CASIN  6614     DIGIN  3221     FATNC7 6326     FORDON 0663
CASOUT 6703     DIGIN1 3222     FATNC8 6331     FORERR 0503
CCNTR  0646     DIGIT  5176     FATNC9 6334     FORLIM 0721
CGET   6651     DIGLUP 6557     FATNSX 6272     FORLIS 5744
CHAIN  7031     DIM    6472     FATNT  6276     FORSTE 0724
CHARNE 0051     DIMFLA 0034     FATNTT 6301     FORVAR 0454
CHAROK 0641     DIVLP  4705     FDIGIT 3335     FOUND  0576
CHECKW 2344     DIVXTE 3341     FDV    7000     FPADD  4456
CHKFIT 6400     DOITNO 1247     FDVI   7400     FPADDR 4304
CHNERR 3362     DONEN  0745     FENTER 4435     FPDIV  4667
CHNG   3023     DONSW  0650     FERR   0746     FPDOIT 4237
CHNMS  3363     DOSCRT 3042     FEXIT  0000     FPFLAG 0154
CHRGET 0417     DOTZER 6576     FEXPC1 6072     FPGOTO 4273
CKBRAK 0344     DPFLAG 3342     FEXPC2 6075     FPJMP  4317
CKCTRZ 4174     DRINTX 3162     FEXPC3 6100     FPJUMP 4274
CLEARV 2464     DSUBTR 0625     FEXPC4 6103     FPLAC  4351
CLEV   0505     DPTI   6657     FEXPC5 6106     FPLOOP 4202
CLOS   0036     DRTICK 7537     FEXPC6 6111     FPMUL  4530
```

Table 6-7 Cassette BASIC Symbol Table (Cont'd)

/PDP-8/E CASSETTE BASIC PAL8-V8 12/27/72 PAGE 51-1

```
FPNOAD 4270    GETOPR 1012    INLOOP 0573    LET    0304
FPOPER 4305    GETUNI 0240    INLUPF 0434    LETDO  0205
FPPGZ  4227    GETVAR 0303    INPLUP 4034    LETTER 3445
FPSKIP 4314    GETWD  0177    INPPTR 4076    LFXLUP 2331
FPSTO  4322    GLOOP  2711    INPSET 0553    LHALF  3067
FPSUB  4453    GNEXT  0607    INPTN  0272    LIMIT  1600
FPT    4200    GOBOTH 0534    INPUT  4007    LINBUF 5551
FPTEMP 4576    GOLIST 5764    INPUTN 7316    LINENO 0054
FPTR   0061    GOSUB  0507    INSERT 2027    LINFIX 2326
FPZDIV 4736    GOTEMP 0055    INSRTS 2025    LIST   3600
FRINP  1243    GOTIT  7207    INT    6434    LISTAL 3610
FRINP1 1261    GOTIT1 7210    INWDTM 4136    LISTLU 3612
FRNDX  5404    GOTO   0521    IONAM  0227    LISTSO 3611
FROUT  1302    GOTOPR 1202    IOUNIT 1301    LIST2  3636
FRSTNE 2101    GOTSS  1071    IPNOPE 4024    LIST3  3655
FSB    5000    GOTSTE 0634    ISDEF2 3511    LIST4  3661
FSBI   5400    GOUT   7244    ISDIG  6532    LIST5  3676
FSEQ   0050    GPTR   0062    ISDIM  1472    LITRAL 3123
FSGE   0100    GRB    7223    ISIT   4564    LJMS   7255
FSGT   0140    GRDELA 7221    ISITDF 0551    LKER   1312
FSINC1 5713    GSBEND 5777    ISITFU 1105    LKERR  0330
FSINC3 5716    GSBPTR 0163    ISITLI 4100    LLSOUT 7400
FSINC4 5721    GSS1   1555    ISIT1  4422    LNOEND 3626
FSINC5 5724    GSS2   1556    ISLIT  4127    LOADED 4123
FSINC6 5727    GTBKLP 1722    ISSOME 1643    LOCCTR 0045
FSINC7 5732    GTEMP  7270    ISUMIN 1010    LOCTEM 0671
FSINM4 5735    GTJMP  0442    ITSDEF 3513    LOCTMP 1677
FSINOK 5657    GTJMP1 0420    ITSOP  3217    LOG    6114
FSINZ  5705    GTPTR  0036    ITSE   3253    LOGACE 6165
FSINZZ 5710    HAF    0437    ITSOP  1220    LOGFWD 6164
FSIN10 5641    HALF   0451    ITSP   3270    LOGOKW 6167
FSLE   0150    HIGHWD 4333    IUNIT  6660    LOOK   0306
FSLT   0110    HLOOP  2677    JBPENT 3707    LOOKER 7310
FSNE   0040    HLOP1  2731    JISDIG 3344    LOOKUP 7002
FSQRX  5407    HLP    4163    JMATCH 2770    LOWLOC 2166
FST    2000    HNDLR  6600    JPUTCH 0763    LPTOUT 7547
FSTI   2400    HPTR   0063    JUSTO  3142    LUKERR 0326
FTANT1 5677    IAMLES 2103    JUSTOF 3152    LUNIT  0332
FTANT2 5702    IANDO  6764    JUSTOP 3155    LUP    3404
FUNTAB 1131    IAO    1257    JUST1  3137    LUPF   0430
FUPRC1 5762    ICASER 5157    JUST2  3141    L4LUP  3664
FWD    0200    ICOUNT 6662    KBDIN  0421    MAYZER 4612
FXXPFX 6023    IERR   6663    KBDINP 7626    MBREAK 7603
GALT   7242    IEXTLC 0235    KBUFEN 6762    MENDLI 0041
GDIM2  1557    IF     0372    KBUFST 6737    MENDPD 2361
GET    0001    IF1    0403    KEPTR  0077    MGOLIS 0720
GETADD 1400    IGNORE 2137    KEYWD  0231    MGSBEN 0527
GETBLK 1700    IMMED  1155    KIGNOR 1676    MINUS  1316
GETCH  7200    IMMEDA 1155    KM200  0072    MLBEGI 0171
GETCHR 0672    IN     3430    KM4004 2170    MLEND  0172
GETCR  0712    INCHN  7403    K5000  2167    MLINBU 0040
GETIT  0275    INDEV  0131    K6201  7005    MNSONE 0736
GETJ   1765    INDEX1 0014    LASERR 6612    MONITR 0100
GETLIN 2602    INDEX2 0015    LBEGIN 5622    MOREDI 6470
GETLRE 2577    INLCTM 4077    LERR   3355    MOREIN 4000
```

Table 6-7 Cassette BASIC Symbol Table (Cont'd)

/PDP-8/E CASSETTE BASIC PAL8-V8 12/27/72 PAGE 51-2

```
MORERD 1620    NOTHER 0437    OV     0013    07600  4345
MOVE   2007    NOTKWD 0305    010    2171    07603  2774
MOVLUP 2122    NOTNOW 1776    01000  3547    07610  5345
MPRINT 0023    NOTSGN 3271    011    0506    07640A 0754
MPY    5321    NOTTXT 2236    0110   2357    07640B 2765
MPYLUP 4552    NOTVAR 1102    012    0067    07673  3306
MQA    7501    NOTX10 5236    0122   2772    077    0140
MQL    7421    NOTYET 6741    013    1562    07700  2610
MSTBF1 4453    NSYMTA 0053    014    2356    07706A 6544
MTXXIT 5173    NTCHAN 1220    0140A  0762    07706C 3472
MULCLR 4571    NTEND  1150    0140B  2745    07715  2775
MULEXP 3367    NULIM  1200    017    5144    07725A 3075
MULXTE 3340    NUMBUF 5335    01740  3352    07725B 3346
MULX1  3371    NUMCHK 0726    0177   0027    07737  3116
MUSTBE 4566    NVULMS 3124    01770  3353    07740  0056
NAM    0322    OADD   4435    02     0064    07741  3743
NAMCHK 0651    OCASER 2574    020    1151    07743  4743
NAMF   1037    OCC    5205    0200   0153    07745  0160
NAMER  0032    OCOUNT 6761    02040A 3350    07753  3101
NAMLOC 0216    OERR   6757    02062  5344    07762  2325
NCTRLZ 6725    OIP    7357    0212   0175    07763A 0761
NEW    1047    OJUMP  1276    0215   0174    07763B 3076
NEWCHA 2622    OLD    1057    023    1366    07764A 1274
NEWDO  1052    OLDDO  1062    0233   3770    07764B 3077
NEWLIN 2606    OLDKLG 2063    0240   0051    07764C 3351
NEWMES 1106    OLDMES 1111    0253   5145    07766  5151
NEXT   0600    OLDOP  0070    0255   5146    07770  0057
NEXTER 0673    OLD3   1100    0256A  5153    07771  5152
NEXTG  2757    ONE    0147    0256B  6575    07772  5154
NEXTVA 0637    ONEDIM 1061    0260   0012    07773  5346
NFMES  0507    ONESS  1073    027    3345    07774  1561
NFOPEN 6365    ONLY1  3312    0305   5147    07776  3347
NM1    1043    OO7600 5452    032    5115    PACN   4742
NORUMP 4633    OO7736 2324    036    2771    PAD    0347
NOCHNG 3040    OPDONE 1203    03734  1152    PADDON 0741
NOCOM  6176    OPE    0030    03737B 0776    PAKBUF 0763
NOCOMM 0327    OPEN   3575    03755  1273    PAL1   0146
NOCR   4056    OPEN1  1200    0377   0073    PANORM 0144
NONBLN 3104    OPERAN 0075    04     0156    PARGER 0047
NONZER 5014    OPNERR 1354    040    2777    PAR1   0145
NOPARE 1032    OPUTAB 7016    04014  1153    PASSCR 0474
NOPCR  2220    OPS    0026    042    3103    PBARRO 2773
NOPE   1314    OPUTC  7147    04200  3102    PBEGFI 1773
NORLFT 6423    OP1    0023    04213  1154    PCCUNT 0744
NORMED 5220    OP2    0022    05400  5347    PCHKFI 0161
NORMIT 5207    OP3    0021    06203  0046    PCLOS  0143
NORUBO 5574    OTEMP  1271    07     0074    PCOMMA 0273
NOSS1  1457    OTEMP1 7012    0700   1272    PCOWT  0137
NOSS2  1452    OTHER  3000    07000A 2565    PDL    0036
NOT    3426    OUNIT  6760    07000B 3473    PDLIST 5703
NOTBAD 2153    OUPSET 1026    07077  1275    PEDIT  0122
NOTBIG 4620    OUTDEV 0133    07520  5150    PENDN  0271
NOTCR  3023    OUTD2  0132    07545  4777    PENDNM 0365
NOTEM  6626    OUTNUM 5000    07570  6456    PERMSY 5111
NOTFUL 6712    OUTOK  1332    07577  4577    PERROR 0101
```

Table 6-7 Cassette BASIC Symbol Table (Cont'd)


```
PERSW  0747    PPXXEQ 1162    P7640  0722    SQLOOP 5435
PEVAL  0103    PRENT  2315    QERROR 1571    SQR    5412
PEXECU 0105    PRESET 0136    QMK    0771    SSERR  1563
PEXP   5776    PRINBL 2302    RBSWCH 0135    SSFIX  4775
PFINDI 0672    PRINCO 2306    RCHR   6763    SSONE  0336
PFIX   0110    PRINEX 2215    READ   1622    SSTWO  0337
PFNERR 5546    PPINHA 2260    READIT 6664    ST     3005
PFPLOO 4575    PRINOU 2227    READLO 0046    STAR   1327
PGETAD 0104    PRINSE 2320    READY  6525    STARTB 3000
PGETBL 0117    PRINT  2172    RELATE 1342    STICKI 6430
PGETCH 0052    PRINTC 2207    REMPAC 3043    STOBUF 0747
PGETLI 0126    PRINTG 2206    REOFER 6700    STOP   2401
PGETLR 3022    PRINTH 2224    RESET1 7072    STORCH 0704
PGETOP 1376    PRINTN 7322    RESTOR 3771    STOVAR 0333
PGETVA 0115    PRINTX 3702    RETNER 0713    SWP    7521
PGOLIS 0162    PRINT2 2174    RETURN 0677    SXER   0551
PGOTOP 0111    PRINUM 3745    RHALF  3071    SXERR  6441
PHLP   0363    PRINVA 3651    RIGHT  0433    TAB    5547
PIGNOR 0473    PRLOOP 3711    RIP    6661    TABDES 6375
PINT   5676    PRN    0243    RMLEFT 6413    TABDO  6350
PISITL 0173    PRUGNA 1114    RND    5353    TABFLG 2343
PLNEGI 0170    PRSUBR 3722    RNDJMP 5350    TABL   3652
PLETDO 0204    PRTEMP 0042    RTBEND 1600    TABTHR 2360
PLETTE 3100    PRTXRE 3720    RTBUF  1400    TAN    5600
PLIMIT 2566    PSAVE  2463    RTIOME 3103    TBEGFI 5572
PLINBU 0037    PSGN   5675    RUBO   5573    TEMP   0003
PLINFI 0157    PSKIP1 1616    RUN    2457    TEMPS  3102
PLIST  2570    PSLOOP 0120    RUNC   2456    TEN    0000
PLITRA 3354    PSPACE 1560    RUNIN  2510    THESKI 1353
PLOG   5775    PSTICK 0123    RUNLUP 2472    TLSOUT 7402
PLUS   1312    PSTOP  3776    RUNNOT 2504    TMP    0031
PMBREA 0364    PSTOVA 0114    RUN2IN 2550    TOOLON 5162
PMPY   5155    PSXERR 0102    RUN2LU 2521    TPRINT 6376
PNBF6  5156    PSYNTA 0052    RUN2NO 2544    TRALUP 2131
PNEWLI 1177    PTABDE 5570    SAVE   1000    TRANSF 2126
PNONBL 0124    PTABFL 5571    SAVE1  2461    TRYAGI 5131
PNOTNO 2573    PTABLE 2776    SCHMOR 1656    TRYAL  0270
PNUMBU 0044    PTEN   0150    SCRATL 2445    TRYALT 0271
POADD  0155    PTEXT  0100    SCRAT1 0332    TRYOLD 3075
POIP   0035    PTUBIG 3021    SEARCH 1657    TRYSTE 0626
POP    4166    PUSERF 5545    SETINC 0200    TTYO   7024
POPERA 3121    PUSH   2362    SETSGN 4512    TTYOUT 7350
POP3   4434    PUTCOF 7014    SGN    0726    TUBIG  1173
POUTNU 0121    PUTCH  0741    SIMPLV 3465    TWIDTH 2355
PPASSC 0112    PUTER  7000    SIN    5624    TWOLF  3557
PPOLIS 0127    PUTJ   3550    SJUMP  0241    TWOSS  1074
PPERMS 2572    PWHERE 0647    SKIPIT 0471    TXTPAK 3046
PPFLOO 4741    PXFORL 0557    SKPSYM 2762    TYQUES 1145
PPFORL 1764    PXLINE 3744    SLASH  1332    UOOPER 1363
PPINT  6060    PXXCRL 3117    SLOOP  2675    UGH1   3561
PPOP   0107    PXXEOF 2571    SLSHTM 1337    UJMS   0004
PPRINT 0116    PXXEXI 3120    SNUMFL 0066    UNOPER 1321
PPRINU 0125    PXXLIT 3122    SPACER 1163    UNDERF 4645
PPUSH  0106    PXXTHE 2567    SPLEFT 0142    UNIT   0012
PPUTCH 0033    PZERDO 6545    SQEXIT 5450    UPARRO 6457
```

Table 6-7 Cassette BASIC Symbol Table (Cont'd)


```
UPARRX 5740     XXGOSU 5321     XXTAN  5204
UPARR2 4365     XXGOTO 5300     XXTEXT 5540
USERFN 1617     XXGT   5147     XXTHEN 5305
UTEM   0411     XXIF   5312     XXTO   5165
UTEM1  0021     XXINPT 5332     XXTTYO 5437
UUJMP  0400     XXINPU 5337     XXUNAR 5542
UUJMS  0401     XXINT  5231     XXUPAR 5122
UUUJMS 0416     XXLBRA 5163     XXUUF  5253
VAR    0335     XXLE   5124     X10    0010
VARTEM 0554     XXLET  5274     ZCNTLO 7147
VSCHIN 3523     XXLIS  5502     ZERDON 5142
VSCHLU 3474     XXLIST 5476     ZERO   0150
VSCHNO 3517     XXLITO 5545     ZFIXEX 4767
WAIT   6200     XXLOG  5212
WAITR  6600     XYLPT  5444
WDTEMP 1072     XXLT   5145
WORD   0050     XYMINU 5114
WTEM   6613     XXNAM  5534
XEXECU 0414     XXNAME 5530
XGISIT 7115     XXNCOM 5447
XGMST1 0533     XYNE   5132
XGMUST 7270     XXNEW  5525
XISIT  7124     XXNEXT 5326
XISJT1 0520     XXNRUB 5467
XMUST  7277     XXOD1  0005
XMUST1 0541     XXOD2  0006
XREG2  0011     XXOD3  0007
XRLOC  0010     XXOLD  5522
XRPUT  0011     XXOPEN 5161
XXABS  5223     XXOPN  5412
XXATN  5207     XXOUTP 5422
XXBSLS 5260     XXPLUS 5112
XXCHAN 5427     XXPRIN 5267
XXCLOS 5155     XXPRNT 5262
XXCLSE 5416     XXPUT  5242
XXCOMA 5455     XXRBRA 5157
XXCOMM 5151     XXREAD 5376
XXCOS  5201     XXREM  5366
XXCRLF 5256     XXRETR 5344
XXDATA 5402     XXRND  5234
XXDEF  5372     XXRSTO 5361
XXDIM  5355     XXRUB  5462
XXEG   5135     XXRUN  5505
XXEL   5140     XXSAV  5517
XXEND  5434     XXSAVE 5513
XXENDN 5406     XXSCR  5510
XXEOF  5541     XXSEMI 5153
XXEQ   5143     XXSGN  5226
XXEXIT 5544     XXSIN  5237
XXEXP  5215     XXSLAS 5120
XXFINI 5543     XXSQR  5220
XXFN   5176     XXSTAR 5116
XXFOR  5315     XXSTEP 5171
XXGE   5127     XXSTOP 5351
XXGET  5245     XXTAB  5250
```

CHAPTER 7
USING CAPS-8 CODT

Using CODT, the programmer can run his binary program on the computer, control its execution, and make alterations to his program by typing at the Teletype keyboard.

CODT occupies any four pages of core, in the same field, that the user desires. The user may change the location of these four pages of core by reassembling the source. If the user program resides in the first few pages of memory, then CODT should be loaded in the upper pages of memory, and vice versa. The user program cannot occupy (overlay) any location used by CODT, including the breakpoint locations (locations 4, 5, and 6 on page zero).

7.1 FEATURES

CODT features include location examination and modification; octal core dumps to the Teletype using the word search mechanism; and instruction breakpoints to return control to CODT (breakpoints). The user's program can run with interrupts on. CODT may reside in any field, not necessarily the same as the user's field.

The breakpoint is one of CODT's most useful features. When debugging a program, it is often desirable to allow the program to run normally up to a predetermined point, at which the programmer may examine and possibly modify the contents of the accumulator (AC) , the link (L), or various instruction or storage locations within his program, depending on the results he finds. To accomplish this, CODT acts as a monitor to the user program.

The user decides how far he wishes the program to run and CODT inserts an instruction in the user's program which, when encountered, causes control to transfer back to CODT. CODT immediately preserves in designated storage locations the contents of the LINK and AC at the break. It then prints out the location at which the break occurred, as well as the contents of the LINK and AC at that point. CODT will then allow examination and modification of any location of the user's program (or those locations containing the AC and L). The user may also move the breakpoint and request that CODT continue running his program. This will cause CODT to restore the AC and L, execute the trapped instruction and continue in the user's program until the breakpoint is again encountered or the program is terminated normally.

7.2 USING CODT

When the programmer is ready to start debugging a new program at the computer, he should have at the console:

1. A binary copy of the new program on a cassette.
2. A complete octal/symbolic program listing.
3. A binary copy of the CODT program (previously assembled so as not to interfere with the user's program).

RUN PROG, CODT

The binary of CODT must be the last file to be run. Execution automatically begins in CODT.

7.2.1 Commands

SLASH(/)--OPEN PRECEDING LOCATION

The location examination character (/) causes the location addressed by the octal number preceding the slash to be opened and its contents printed in octal. The open location can then be modified by typing the desired octal number and closing the location. Any octal number from 1 to 4 digits in length is a legal input. Typing a fifth digit is an error and will cause the entire modification to be ignored and a question mark to be printed by CODT. Typing / with no preceding argument causes the latest named location to be opened (again). Typing 0/ is interpreted as / with no argument. For example:

```
400/6046
400/6046 2468?
400/6046 12345?
/6046
```

The memory field referenced is that field specified by the location in CODT symbolically referenced by F. For example, if the contents of F were 20, then the command

```
423/
```

would examine location 423 in memory field 2.

RETURN--CLOSE LOCATION

If the user has typed a valid octal number, after the content of a location is printed by CODT, typing the RETURN key causes the binary value of that number to replace the original contents of the opened location and the location to be closed. If nothing has been typed by the user, the location is closed but the content of the location is not changed. For example:

```
400/6046            location 400 is unchanged
400/6046 2345       location 400 is changed to contain 2345.
/2345 6046          replace 6046 in location 400.
```

Typing another command will also close an opened register. For example:

```
400/6046 401/6031 2346   location 400 is closed and unchanged and
400/6046 401/2346        401 is opened and changed to 2346.
```

LINE FEED--CLOSE LOCATION, OPEN NEXT LOCATION

The LINE FEED key has the same effect as the RETURN KEY, but, in addition, the next sequential location is opened and its contents printed. For example:

```
400/6046            location 400 is closed unchanged and 401
0401/6031 1234      is opened. User types change, 401 is
0402/5201           closed containing 1234 and 402 is
                    opened.
```

SEMICOLON--CLOSE LOCATION, AND UNOBTRUSIVELY OPEN NEXT LOCATION

The SEMICOLON key has the same effect as the LINEFEED key except that the location and contents of the next sequential location are not printed. Therefore,

```
400/6046 3211; 4162; 5000
```

has the same effect as

```
400/6046 3211
401/6031 4162
402/5201 5000
```

This makes it convenient for the user to change several sequential locations.

↑ (SHIFT/N)--CLOSE LOCATION, TAKE CONTENTS AS MEMORY REFERENCE AND OPEN SAME

The up arrow (circumflex) will close an open location just as will the RETURN key. Further, it will interpret the contents of the location as a memory reference instruction, open the location referenced and print its contents. For example:

```
404/3270↑           3270 symbolically is "DCA, this page,
0470/0212 0000      relative location 70," so CODT opens
                    location 470.
```

← (SHIFT/O) CLOSE LOCATION, OPEN INDIRECTLY

The Back arrow (or underscore) will close the currently open location and then interpret its contents as the address of the location whose contents it is to print and open for modification. For example:

```
365/5760↑
0360/0426←
0426/5201
```

nnnnG--TRANSFER CONTROL TO USER AT LOCATION nnnn

Clear the AC then go to the location specified before the G (in the field specified by F). All indicators and registers will be initialized and the breakpoint, if any, will be inserted. Typing G alone will cause a jump to location Ø.

nnnnB--SET BREAKPOINT AT USER LOCATION nnnn

Instructs CODT to establish a breakpoint at the location specified before the B (in the field specified by F). If B is typed alone, CODT removes any previously established breakpoint and restores the original contents of the break location. A breakpoint may be changed to another location, whenever CODT is in control, by simply typing nnnnB where nnnn is the new location. Only one breakpoint may be in effect at one time; therefore, requesting a new breakpoint removes any previously existing one.

A restriction in this regard is that a breakpoint may not be set on any of the floating-point instructions which appear as arguments of a JMS. For example:

```
TAD
DCA     Breakpoint legal here.
JMS
FADD    Breakpoint illegal here.
```

A breakpoint may not be set on a CIF instruction, nor on an instruction which is meant to be executed between a CIF and its corresponding JMP or JMS instruction.

A breakpoint may not be set on a memory reference instruction which references an auto-index register.

A breakpoint may not be set on a two-word EAE instruction, nor may it be set on any of the following instructions:

SKON
ION
IOF

The breakpoint (B) command does not make the actual exchange of CODT instruction for user instruction, it only sets up the mechanism for doing so. The actual exchange does not occur until a "go to" or a "proceed from breakpoint" command is executed.

When, during execution, the user's program encounters the location containing the breakpoint, control passes immediately to CODT (via location 4, 5, and 6). The C(AC) and C(L) at the point of interruption are saved in special locations accessible to CODT. The user's data field is stored in location D and his instruction field is stored in location F as well as internally by CODT. The user instruction that the breakpoint was replacing is restored before the address of the trap and the content of the LINK and AC are printed. The restored instruction has not been executed at this time. It will not be executed until the "proceed from breakpoint" command is given. Any user location, including those containing the stored AC and Link, can now be modified in the usual manner. The breakpoint can also be moved or removed at this time.

An example of breakpoint usage follows the section "CONTINUE AND ITERATE LOOP...".

A--OPEN C(AC)

When the breakpoint is encountered the C(AC) and C(L) are saved for later restoration. Typing A after having encountered a breakpoint opens for modification the location in which the AC was saved and prints its contents. This location may now be modified in the normal manner (see Slash) and the modification will be restored to the AC when the "proceed from breakpoint" command is given.

Similarly, other key locations in CODT may be examined and modified as follows:

L--OPEN CONTENTS OF LOCATION L (LINK)

F--OPEN CONTENTS OF LOCATION F

D--OPEN CONTENTS OF LOCATION D (USER'S DATA FIELD)

NOTE

Whenever any of the locations A, L, F, D, M are referenced, CODT automatically sets the value of F to be the field of CODT.

C--PROCEED (CONTINUE) FROM A BREAKPOINT

Typing C, after having encountered a breakpoint, causes CODT to insert the latest specified breakpoint (if any); restore the contents of the AC and Link; execute the instruction trapped by the previous breakpoint; and transfer control back to the user program at the appropriate location. The user program then runs until the breakpoint is again encountered.

Regardless of the value of F, the C command resumes program execution at the precise spot where it had been previously stopped. The user's data field is first set to that specified by location D.

NOTE

If a trap set by CODT is not encountered while CODT is running the object (user's) program, the instruction which causes the break to occur will not be removed from the user's program.

nnnnC--CONTINUE AND ITERATE LOOP nnnn TIMES BEFORE BREAK

The programmer may wish to establish the breakpoint at some location within a loop of his program. Since loops often run to many iterations, some means must be available to prevent a break from occurring each time the break location is encountered. This is the function of nnnnC (where nnnn is an octal number). After having encountered the breakpoint for the first time, this command specifies how many additional times the loop is to be iterated before another break is to occur. The break operations have been described previously in the section on the B command.

Given the following program, which increases the value of the AC by increments of 1, the use of the breakpoint command may be illustrated.

```
            *200
0200 7300       CLA CLL
0201 1206   A,  TAD ONE
0202 2207   B,  ISZ CNT
0203 5202       JMP B
0204 5201       JMP A
0205 7402       HLT
0206 0001   ONE,1
0207 0000   CNT,0
A       0201
B       0202
CNT     0207
ONE     0206

0201B
200G
0201 (0;0000
C
0201 (0;0001
C
0201 (0;0002
4C
0201 (0;0007
```

CODT has been loaded and started. A breakpoint is inserted at location 0201 and execution stops here showing the AC initially set to 0000 and the link 0. The use of the Proceed command (C) executes the program until the breakpoint is again encountered (after one complete loop) and shows the AC to contain a value of 0001. Again execution continues, incrementing the AC to 0002. At this point, the command 4C is used, allowing execution of the loop to continue 4 more times (following the initial encounter) before stopping at the breakpoint. The contents of the AC have now been incremented to 0007.

M--OPEN SEARCH MASK

Typing M causes CODT to open for modification the location containing the current value of the search mask and print its contents. Initially the mask is set to 7777. It may be changed by opening the mask location and typing the desired value after the value printed by CODT then closing the location.

LINE FEED--OPEN LOWER SEARCH LIMIT

The word immediately following the mask storage location contains the location at which the search is to begin. Typing the LINE FEED will open for modification the first location after the mask, and its contents will be printed. Initially, the lower search limit is set to 0001. It may be changed by typing the desired lower limit after that printed by CODT, then closing the location.

LINE FEED--OPEN UPPER SEARCH LIMIT

The next sequential word contains the location with which the search is to terminate. Typing the LINE FEED key to close the lower search limit causes the upper search limit to be opened for modification and its contents printed. Initially, the upper search limit is the beginning of CODT itself, 7000 (1000 for low version). It may also be changed by typing the desired upper search limit after the one printed by CODT, then closing the location with the RETURN key.

nnnnW--WORD SEARCH

The command nnnnW (where nnnn is an octal number) will cause CODT to conduct a search of a defined section of core, using the mask and the lower and upper limits which the user has specified, as indicated above. The searching operations are used to determine if a given quantity is present in any of the locations of a particular section of memory.

The search is conducted as follows: CODT masks the expression nnnn which the user types preceding the W and saves the result as the quantity for which it is searching. CODT then masks each location within the user's specified limits with C(M) and compares the result to the quantity for which it is searching. If the two quantities are identical, the address and the actual unmasked contents of the matching location are printed and the search continues until the upper limit is reached. The search occurs in the memory field specified by F.

A search never alters the contents of any location. For example: Search location 3000 to 4000 for all ISZ instructions regardless of what location they refer to (i.e. search for all locations beginning with an octal 2).

F0010	20	Set the field to 2
M7777	7000	Change the mask to 7000, open lower search limit.
7453/0001	3000	Change the lower limit to 3000, open upper limit.
7454/7000	4000	Change the upper limit to 4000, close location.
2000W		Initiate the search for ISZ instructions.
0200/2467		
3057/2501		These are 4 ISZ instructions in
3124/2032		this section of core.

7.3 ILLEGAL CHARACTERS

Any character that is neither a valid control character nor an octal digit, or is the fifth octal digit in a sequence, causes the current line to be ignored and a question mark printed. For example:

```
4:?                 CODT opens no location.
4U?
0406/4671 67K       CODT ignores modification and closes
/4671               location 406.
```

7.4 ADDITIONAL TECHNIQUES

7.4.1 TTY I/O-FLAG

CODT automatically notes the status of the TTY flag after encountering a breakpoint and restores it after performing a CONTINUE.

7.4.2 Interrupt Program Debugging

CODT executes an IOF when a breakpoint is encountered. (It does not do this when more iterations remain in an nnnnC command.) This is done so that an interrupt will not occur when CODT prints the breakpoint information. CODT thus protects itself against spurious interrupts and may be used safely in debugging programs that turn on the interrupt mode. CODT restores the interrupt facility to its former state when it resumes execution.

7.4.3 Octal Dump

By setting the search mask to zero and typing W, all locations (in field F) between the search limits will be printed on the Teletype.

7.4.4 Indirect References

When an indirect memory reference instruction is encountered, the actual address may be opened by typing ↑ and ← (SHIFT/N and SHIFT/O, respectively).

7.5 ERRORS

The only legal inputs are control characters and octal digits. Any other character will cause the character or line to be ignored and a question mark to be printed by CODT. Typing G alone is an error. It must be preceded by an address to which control will be transferred This will elicit no question mark also if not preceded by an address, but will cause control be transferred to location 0.

7.6 OPERATION AND STORAGE

7.6.1 Storage Requirements - CAPS-8 System

CODT can be run in a standard CAPS-8 system and requires 1000 (octal) core locations and three locations (4,5,6) on page zero of every field which uses a breakpoint. CODT is page-relocatable.

The source tape can be re-origined to the start of any memory page except page zero and assembled to reside in the four pages following that location, assuming they are all in the same memory bank.

7.6.2 Programming Notes Summary

CODT must begin at the start of a memory page (other than page zero) and must be completely contained in one memory field.

The user's program must not use or reference any core locations occupied or used by CODT and vice versa.

Breakpoints are fully invisible to "open location" commands; however, breakpoints may not be placed in locations which the user program will modify in the course of execution or the breakpoint will be destroyed.

If a trap set by CODT is not encountered by the user's program, the breakpoint instruction will not be removed.

The user may type CTRL/C at any time to return to the CAPS-8 Monitor (assuming his program did not destroy CAPS-8).

7.7 COMMAND SUMMARY

nnnn/	Open location designated by the octal number nnnn.
/	Reopen latest opened location.
RETURN key	Close previously opened location.
LINE FEED key	Close location and open the next sequential one for modification.
;	Close location and allow immediate modification of the next location.
↑(SHIFT/N)	Close location, take contents of that location as a memory reference and open it.
←(SHIFT/O)	Close location, open indirectly.
Illegal character	Current line typed by user is ignored, CODT types: ?(CR/LF).
nnnnG	Transfer program control to location nnnn.
nnnnB	Establish a breakpoint at location nnnn.
B	Remove the breakpoint.
A	Open for modification the location in which the contents of the AC were stored when the breakpoint was encountered.
L	Open the location storing the link.
C	Proceed from a breakpoint.
nnnnC	Continue from a breakpoint and iterate past the breakpoint nnnn times before interrupting the user's program at the breakpoint location.
M	Open the search mask.
LINE FEED key	Open lower search limit.
LINE FEED key	Open upper search limit.
nnnnW	Search the portion of core as defined by the upper and lower limits for the octal value nnnn.
F	Open location F.
D	Open location D.

CHAPTER 8
CAPS-8 UTILITY PROGRAM

8.1 INTRODUCTION

The CAPS-8 Utility Program (UTIL) allows the user to take files stored on paper tape and transfer them to cassette, using either the high-speed or low-speed reader. The Utility Program will transfer both ASCII and BINARY files.

8.2 CALLING AND USING THE UTILITY PROGRAM

To call the Utility Program from the system cassette, the user types:

```
.R UTIL/OPTIONS
```

in response to the dot (.) printed by the Keyboard Monitor.

8.2.1 Utility Program Options

There are two options available for use with the Utility Program; these options are discussed in Table 8-1. (Options usage is explained in Chapter 2, Section 2.4.3.)

Table 8-1
Utility Program Options

Option	Meaning
/H	The /H option is used to designate the high-speed reader as the input device. Note that the high-speed reader is the default input device. That is, if no option is specified the program assumes that the high-speed reader is the input device.
/L	The /L option is used to designate the low-speed reader as the input device.

8.2.2 Input and Output Specifications

Before indicating the input and output devices to be used in the file transfer, the user must ensure that the proper reader is ready and that the cassette the file is to be copied onto is write-enabled. When this has been done, the user is ready to begin the file transfer.

After UTIL has been called from the system cassette, it asks the following:

```
MODE-
```

The user responds with a single character (A or B) to indicate whether the file being put on the cassette is ASCII or BINARY.

```
OUT-
```

The user responds by typing the drive number the output cassette is mounted on and the name of the output file to be put onto it, i.e., 3:FOO. In B mode, .BIN is the default extension.

After these two queries have been answered UTIL prints the following:

```
↑
```

The user responds by typing any character; this response starts the file transfer. (If /L had been typed, the user merely turns on the low-speed reader.)

8.3 UTILITY PROGRAM ERROR MESSAGES

Errors which occur during the Utility Program operation may be of two types: User errors and cassette errors. User errors may be corrected with the appropriate action as detailed in Table 8-2. Cassette errors normally require the user to use another cassette to complete the copy operation. Control does not return to the Keyboard Monitor when a Utility Program error occurs. The user may use CTRL/C to return to the Monitor if he cannot correct the indicated error.

Table 8-2
Utility Program Error Messages

Message	Meaning
UNIT n NOT READY	There is either no cassette on the cassette drive specified or no such drive exists.
OUTPUT ERROR ON UNIT n	An output error has occurred on the cassette drive specified. The user should try the transfer operation using another cassette. Or perhaps, the user tried to write data when the write protect tab of the cassette on the drive specified was write-locked.

ENTER ERROR ON UNIT n	An error occurred while trying to open a new cassette file.
UNIT n FULL	There was not enough room on the cassette.
CLOSE ERROR ON UNIT n	An error occurred during a close operation.
INPUT ERROR	In binary mode, the paper tape reader stopped or ran out of tape before a checksum was encountered.
CHECKSUM ERROR	In binary mode, the checksum did not agree; probably a hardware read error. Try again.

Whenever an error occurs, the program writes a new sentinel on the open cassette if possible.

CHAPTER 9

BOOT

BOOT is used to make it convenient to bootstrap from one system to another, or from one device to another by typing commands on the keyboard.

BOOT can run conveniently from CAPS-8 and other monitor systems, i.e., OS/8 and COS-300.

9.1 OPERATING PROCEDURES

To run BOOT from CAPS-8 the user types:

```
.R BOOT
```

the system will respond by typing a slash, at which time the user responds with the device mnemonic.

If an illegal mnemonic is typed, the system types "NO" and prints a slash to allow the user to try again. In this case, the user can type RUBOUT to erase a line and try again.

If a legal mnemonic was given, but the system configuration does not include the corresponding device (or the device is not ready), the bootstrap may hang.

The following is a list of legal mnemonics:

Mnemonic	Device	System or Comments
CA	TA8E cassette	CAPS-8
DK	Any disk (RF08, DF32, RK8E, RK8)	OS/8, COS-300
DL	LINCtape	DIAL-V2, DIAL-MS
DM	RF08 or DF32	Disk Monitor
DT	Any tape (TC08, TD8E, LINCtape)	OS/8, COS-300
LT	LINCtape	OS/8, COS-300
PT	PT8E Papertape	Loads BINLDR into field 0

Mnemonic	Device	System or Comments
RE	RK8E disk	OS/8, COS-300
RF	RF08, DF32 disks	OS/8, COS-300
RK	RK8 disk	OS/8, COS-300
TD	TD8E DECtape	OS/8, COS-300
TY	TC08 DECtape unit 4	Typeset Bootstrap types BOOT's version number
TC	TC08 DECtape	OS/8, COS-300, Disk monitor, DEC library system, and others
ZE		Zeroes core (field 0)

If the device mnemonic is followed by a period, the program will load the correct bootstrap into core and then halt. Hitting continue branches to the bootstrap.

Example:

```
.R BOOT
/DT
```

The preceding bootstraps onto a DECtape system (on DECtape unit Ø). The underlined characters were typed by the computer.

APPENDIX A
ASCII CHARACTER SET

Character	8-Bit Octal	7-Bit Octal	Character	8-Bit Octal	7-Bit Octal
A	301	101	!	241	241
B	302	102	"	242	242
C	303	103	#	243	243
D	304	104	$	244	244
E	305	105	%	245	245
F	306	106	&	246	246
G	307	107	'	247	247
H	310	110	(	250	250
I	311	111	)	251	251
J	312	112	*	252	252
K	313	113	+	253	253
L	314	114	,	254	254
M	315	115	-	255	255
N	316	116	.	256	256
O	317	117	/	257	257
P	320	120	:	272	272
Q	321	121	;	273	273
R	322	122	<	274	274
S	323	123	=	275	275
T	324	124	>	276	276
U	325	125	?	277	277
V	326	126	@	300	
W	327	127	[	333	133
X	330	130	\	334	134
Y	331	131	]	335	135
Z	332	132	↑	336	136
0	260	260	←	337	137
1	261	261	Leader/Trailer	200	
2	262	262	BELL	207	
3	263	263	TAB	211	
4	264	264	LINE FEED	212	
5	265	265	FORM	214	
6	266	266	CARRIAGE RETURN	215	
7	267	267	CTRL/Z	232	
8	270	270	SPACE	240	240
9	271	271	RUBOUT	377	
			BLANK	000	

APPENDIX B
ERROR MESSAGE AND COMMAND SUMMARIES

The following summaries are provided for the user's convenience; they are grouped in alphabetical order according to the System Program to which they pertain. As these are only summaries the user is referred to the appropriate chapters for details.

KEYBOARD MONITOR (Chapter 2)

Error Messages

Message	Meaning
BAD COMMAND	The user has failed to follow the correct syntax for Monitor commands.
FILE NOT FOUND	The Monitor could not locate the file (or files) specified.
IO ERROR ON UNIT n	An I/O error has occurred on the cassette unit drive specified. The user should try the I/O transfer specifying another cassette.
UNIT n NOT READY	There is no cassette on the unit drive specified, or no such drive exists.
UNIT n WRITE LOCKED	The user tried to write data when the write protect tab of the cassette on the drive specified was write-locked.

Commands

Command	Explanation
DATE	Allows the user to enter the month, day, and year. This date is then represented in directory listings.
DELETE	Causes the file named in the command line to be deleted from the cassette drive specified.
DIRECTORY	Causes a directory listing of the cassette specified in the command line.

KEYBOARD MONITOR (Con't)

Command	Explanation
LOAD	Instructs the Monitor to load the file(s) specified in the command line. (The correct starting address is then set in the switch register and execution is started by pressing CONTinue.)
REWIND	Causes the cassette on the drive specified to be rewound to its beginning.
RUN	Instructs the Monitor to load and execute the file(s) specified in the command line.
VERSION	Causes the version number of the Monitor currently in use to be printed on the console terminal.
ZERO	Causes deletion of all files following the filename specified in the command line. If no filename is indicated, all files are deleted and the sentinel file is moved to the beginning of the cassette.

EDITOR (Chapter 3)

Error Messages

Error codes are printed in the form ?n↑C where n represents one of the following:

Code	Meaning
0	The EDITOR failed in reading from a device. An error occurred in the device handler; most likely a hardware malfunction.
1	The EDITOR failed in writing onto a device; generally a hardware malfunction.
2	A file close error occurred. The output file could not be closed.

A question mark (?) may appear any time the EDITOR encounters a syntax error.

In addition, the EDITOR contains the following error message:

Message	Meaning
UNIT HAS OPENED FILE	Two files cannot be open on the same cassette at the same time.

Commands

Command	Meaning
A	Append text from the keyboard to whatever is present in the text buffer.
B	List the number of available memory locations in the text buffer.
C	Change the text of a specified line or lines.
D	Delete the specified line(s) from the buffer.
E	Output the current buffer and transfer all input to the output file; close the output file.
F	Find the next occurrence of the string currently being searched for.
G	Get and list the next line which has a label associated with it.
I	Insert text before a specified line in the text buffer.
J	Initiate an inter-buffer search for a character string.
K	Kill the buffer; reset the text buffer pointers so that there is no text in the buffer.
L	List entire contents (or specified lines) of the text buffer on console terminal.
M	Move specified lines from one place in the text to another, deleting the old occurrence of the text.
N	Write the current buffer to the indicated output file, kill the buffer and read the next logical page from the input file.

EDITOR (Con't)

Command	Meaning
P	Write the entire text buffer (or specified lines) to the output buffer.
Q	Immediate end-of-file. Q causes the text buffer to be output to the output file and the file closed.
R	Read from the specified input device and append the new text to the current contents of the buffer.
S	Search for the character specified.
V	List the entire text buffer (or only specific lines) on the line printer.
Y	Skip to a logical page in the input file, without writing any output.
$	Perform a search for a specified string of characters.
.= or .: /= or /:	List the current line number (.=) or list the last line number in the text buffer (/=).
>	List the next line in the text buffer on the console terminal.
<	List the previous line in the text buffer on the console terminal.
LINE FEED Key	List the next line in the text buffer on the console terminal.

SYSCOP (Chapter 4)

Error Messages

Message	Meaning
INPUT ERROR ON UNIT n	An input error has occurred on the cassette unit specified. The user should try the copy operation using another cassette.
UNIT n NOT READY	There is no cassette on the unit drive specified, or no such drive exists.

SYSCOP (Con't)

Message	Meaning
UNIT n WRITE LOCKED	The user tried to write data when the write protect tab of the cassette on the drive specified was write-locked.
OUTPUT ERROR ON UNIT n	An output error has occurred on the cassette unit specified. The user should try the copy operation using another cassette.

PALC (Chapter 5)

Error Messages

Error Code	Explanation
BE	Two PALC Internal tables have overlapped. Fatal error.
DE	An error was detected when trying to read or write a device.
DF	Device full.
IC	Illegal character. The character is ignored and assembly continues.
ID	Illegal redefinition of a symbol. The symbol retains its old definition.
IE	Illegal equals. An equal sign was used in the wrong context.
II	Illegal indirect. An off-page reference was made.
IP	Illegal pseudo-op. A pseudo-op was used in the wrong context or with incorrect syntax.
IZ	Illegal page zero reference. The pseudo-op Z was found in an instruction which did nct refer to page zero. The Z is igncred.
PE	Current non-zero page exceeded. An attempt was made to: 1. Override a literal with an instruction

PALC (Con't)

Error Code	Meaning
	2. Override an instruction with a literal
	3. Use more literals than the assembler allows on that page
PH	Phase error. Either no $ appeared at the end of the program, or < and > in conditional pseudo-ops did not match.
RD	Redefinition. A permanent symbol has been defined with =. The new and old definitions do not match. The redefinition is allowed.
SE	Symbol table exceeded. Too many symbols have been defined for the amount of memory available.
UO	Undefined origin. An undefined symbol has occurred in an origin statement.
US	Undefined symbols. A symbol has been processed during pass 2 that was not defined before the end of pass 1.
ZE	Page 0 exceeded. Same as PE except with reference to page 0.

BASIC (Chapter 6)

Error Messages

Message	Meaning
ARGUMENT ERROR	A function was given an illegal argument; for example: SQR(-1).
CHAIN ERROR	A cassette error occurred while doing program chaining.
DATA ERROR	There were no more items in the data list.
EOF ERROR	The user attempted to read past the end-of-file during run-time input.
EXPRESSION ERROR	One of BASIC's internal lists overflowed while attempting to evaluate an expression.

BASIC (Con't)

Message	Meaning
FILE NAME ERROR	A mistake was found in the user's specification of a cassette drive # or filename in either a CHAIN or an OPEN statement.
FILE OPEN ERROR	The user attempted to open a run-time output file when one was already open, or a hardware error occurred.
FOR ERROR	FOR loops were nested too deeply.
FUNCTION ERROR	The user attempted to call a function which had not been defined.
GOSUB ERROR	Subroutines were nested too deeply.
I O ERROR	The user attempted to do run-time input and output to the same cassette at the same time.
IN ERROR	A cassette error occurred while attempting to carry out an OLD command or while doing run-time input.
LINE TOO LONG	A line greater than 80 characters in length was typed; BASIC ignores the line and waits for a new one to be entered.
LINE # ERROR	A GOTO, GOSUB, or IF statement referenced a nonexistent line.
LOOKUP ERROR	BASIC could not find a run-time input file on the drive specified.
NEXT ERROR	FOR and NEXT statements were not properly paired.
NO FILES ERROR	The user attempted to do run-time file I/O without first specifying so during BASIC's initial dialogue.
OUT ERROR	An error (probably end-of-tape) occurred while doing cassette output either during a SAVE or during run-time output.
RETURN ERROR	A RETURN statement was issued when not under control of a GOSUB.

Message	Meaning
SUBSCRIPT ERROR	A subscript was used which was outside the bounds defined in the DIM statement.
SYNTAX ERROR	A command did not correspond to the language syntax, or an undefined UUF was referenced.
TOO BIG, LINE IGNORED	The combination of program size and number of variables exceeded the capacity of the computer.

Statements

Statement	Meaning
CHAIN	Link to next user program.
CLOSE	Close open output file.
COMMAS	Output data values to a cassette inserting a comma between each value.
DATA	Set values for a READ.
DEF	Define a function.
DIM	Dimension subscripted variables.
END	Signals the end of program execution.
FOR-TO-STEP	Set up a program loop; increment the counter by a value specified using STEP.
GOSUB	Transfer control to a subroutine.
GOTO	Transfer control to the line number specified in the command line.
IF-END#-THEN	Transfer control (or perform an operation) depending upon the validity of the last INPUT# statement.
IF-GOTO IF-THEN	Transfer control (or perform an operation) depending upon the relationship between variables specified in the command line.

Statement	Meaning
INPUT	Input values from the console terminal.
INPUT#	Input values from a data file.
LET	Assign a value to a variable.
LIST	List program (or specific lines) on console terminal.
LIST#	List program (or specific lines) on line printer.
LPT	Send output to the line printer.
NAME	Rename the program in memory.
NEW	Specify a new program name.
NEXT	Continue a program loop until a terminating value is reached.
NO COMMAS	Terminate outputting of commas.
NO RUBOUT	Disable the RUBOUT command.
OLD	Call saved program from cassette into memory.
OPEN FOR INPUT/OUTPUT	Open a file on cassette for input or output.
PRINT	Print values or specified text on the console terminal.
PRINT#	Output values to a data file.
READ	Read values from a data list.
REM	Insert remarks or comments in the program.
RESTORE	Reset DATA value to its original value.
RETURN	Return from a subroutine to the main body of the program.
RUBOUTS	Allow the use of the RUBOUT key to delete characters.
SAVE	Save the program in memory on the cassette to be specified.

BASIC (Con't)

Statement	Meaning
SCR	Delete the current program in memory.
STOP	Transfer control to the END statement.
TTY OUT	Return output to the console terminal (after using LPT).

Functions

Function	Meaning
SIN(x)	Sine of x
COS(x)	Cosign of x
TAN(x)	Tangent of x
ATN(x)	Arctangent of x
EXP(x)	Exponential value of x
LOG(x)	Natural log of x
SGN(x)	Sign of x
INT(x)	Integer value of x
ABS(x)	Absolute value of x
SQR(x)	Square root of x
RND(x)	Generate a random number
TAB(x)	Print character at space x
GET(x)	Get character from input device
PUT(x)	Put character on output device
FNA(x)	User-defined function
UUF(x)	User-coded function

APPENDIX C
PALC PERMANENT SYMBOL TABLE

The following are the most commonly used elements of the PDP-8 instruction set and are found in the permanent symbol table within the PALC assembler. For additional information on these instructions and for a description of the symbols used when programming other, optional, I/O devices, see THE SMALL COMPUTER HANDBOOK, available from the DEC Software Distribution Center. (All times are in microseconds and respresentative of the PDP-8/E.)

Mnemonic	Code	Operation	Time
Memory Reference Instructions			
AND	0000	Logical AND	2.6
TAD	1000	Two's complement add	2.6
ISZ	2000	Increment and skip if zero	2.6
DCA	3000	Deposit and clear AC	2.6
JMS	4000	Jump to subroutine	2.6
JMP	5000	Jump	1.2

Mnemonic	Code	Operation	Sequence
Group 1 Operate Microinstructions (1 cycle = 1.2 microseconds)			
NOP	7000	No operation	--
IAC	7001	Increment AC	3
RAL	7004	Rotate AC and link left one	4
RTL	7006	Rotate AC and link left two	4
RAR	7010	Rotate AC and link right one	4
RTR	7012	Rotate AC and link right two	4
CML	7020	Complement link	2
CMA	7040	Complement AC	2
CLL	7100	Clear link	1
CLA	7200	Clear AC	1
BSW	7002	Swap Bytes in AC	4

Mnemonic	Code	Operation	Sequence
Group 2 Operate Microinstructions (1 cycle)			
HLT	7402	Halts the computer	3
OSR	7404	Inclusive OR SR with AC	3
SKP	7410	Skip unconditionally	1
SNL	7420	Skip on nonzero link	1
SZL	7430	Skip on zero link	1
SZA	7440	Skip on zero AC	1
SNA	7450	Skip on nonzero AC	1
SMA	7500	Skip on minus AC	1
SPA	7510	Skip on positive AC (zero is positive)	1
Combined Operate Microinstructions			
CIA	7041	Complement and increment AC	2,3
STL	7120	Set link to 1	1,2
GLK	7204	Get link (put link in AC, bit 11)	1,4
STA	7240	Set AC to -1	2
LAS	7604	Load AC with SR	2,3
MQ Microinstructions			
MQL	7421	Load MQ from AC, then clear AC	
MQA	7501	Inclusive OR the MQ with AC	
CAM	7621	Clear AC and MQ	
SWP	7521	Swap AC and MQ	
ACL	7701	Load MQ into AC	
Internal IOT Microinstructions			
SKON	6000	Skip if interrupt ON, and turn OFF	
ION	6001	Turn interrupt processor on	
IOF	6002	Disable interrupt processor	
SRQ	6003	Skip on interrupt request	
GTF	6004	Get interrupt flags	
RTF	6005	Restore interrupt flags	
SGT	6006	Skip on greater than flag	
CAF	6007	Clear all flags	

Mnemonic	Code	Operation
Keyboard/Reader (1 cycle)		
KSF	6031	Skip on keyboard/reader flag
KCC	6032	Clear keyboard/reader flag and AC; set reader run
KRS	6034	Read keyboard/reader buffer (static)
KRB	6036	Clear AC, read keyboard buffer (dynamic), clear keyboard flags
KCF	6030	Clear keyboard/reader
KIE	6035	AC 11 to keyboard/reader interrupt enable F.F.
Teleprinter/Punch (1 cycle)		
TSF	6041	Skip on teleprinter/punch flag
TCF	6042	Clear teleprinter/punch flag
TPC	6044	Load teleprinter/punch and print
TLS	6046	Load teleprinter/punch, print, and clear teleprinter/punch flag
TFL	6040	Set teleprinter/punch flag
TSK	6045	Skip on printer or keyboard flag
Line Printer (1 cycle)		
LSF	6661	Skip on character flag
LCF	6662	Clear character flag
LSE	6663	Skip on error
LPC	6664	Load printer buffer; print on full buffer or control character
LIE	6665	Set program interrupt flag
LLS	6666	Clear line printer flag, load character and print
LIF	6667	Clear program interrupt flag
Cassette (1 cycle)		
KCLR	6700	Clear all
KSDR	6701	Skip on data flag
KSEN	6702	Skip on error
KSBF	6703	Skip on ready flag
KLSA	6704	Load status A from AC 4-11, clear AC, load 8-bit complement of status A
KSAF	6705	Skip on any flag or error
KGOA	6706	Assert the contents of status A, transfer data if read or write
KRSB	6707	Read status B into AC 4-11

Mnemonic	Code	Operation
Memory Extension Control, Type MC8/E (1 cycle)		
CDF	62N1	Change to data field N
CIF	62N2	Change to instruction field N
RDF	6214	Read data field
RIF	6224	Read instruction field
RIB	6234	Read interrupt buffer
RMF	6244	Restore memory field
CDI	62N3	Change to data field and instruction field N

PSEUDO-OPERATORS

The following is a summary of the PALC assembler pseudo-operators and a brief description of their functions. Detailed information concerning these pseudo-ops is contained in Chapter 5.

- DECIMAL - Causes all following numbers to be interpreted as decimal.
- OCTAL - Causes all following numbers to be interpreted as octal.
- FIELD - Causes a field setting.
- I - Represents indirect addressing.
- Z - Denotes a page zero reference.
- EXPUNGE - Deletes the entire permanent symbol table.
- FIXTAB - Appends presently defined symbols to the permanent symbol table.
- PAGE - Resets the location counter to the next page.
- XLIST - Suppresses listing while continuing assembly; a second XLIST continues listing.
- IFDEF - If the symbol is defined, the code within brackets is assembled.
- IFNDEF - If the symbol is not defined, the code within brackets is not assembled.
- IFZERO - If the expression is zero, the code within brackets is assembled.
- IFNZRO - If the expression is not zero, the code within brackets is not assembled.
- FIXMRI - Defines a memory reference instruction.
- ENPUNCH - Resumes binary output after NOPUNCH.
- NOPUNCH - Continues assembling code but stops binary output.
- ZBLOCK - Reserves words of memory.
- EJECT - Causes the listing to jump to the top of the next page.
- TEXT - Allows a string of text characters to be entered.

APPENDIX D
SYSTEM DEMONSTRATION RUN

The following example run, in which the user creates a binary and listing file from an ASCII source file, illustrates a typical use of the CAPS-8 System. The machine output is coded by letters in the left margin which correspond to the textual explanations found following the run.

A
```
.DA 01/04/7 3

.DI 1

01/04/73
FILE   .BIN
MATH   .DAT   12/17/72   V2

.Z 1
```

B
```
.R PALC

-INPUT FILES
*2:TEST.PAL
*
-BINARY FILE
*
-LIST TO
*LPT
US  START
US  L260  +0001
```

C
```
KCLR=6700                                   PALC=V1  01/04/73  PAGE 1

        6700  KCLR=6700
        6701  KSDR=6701
        6702  KSEN=6702
        6703  KSBF=6703
        6704  KLSA=6704
        6705  KSAF=6705
        6706  KGOA=6706
        6707  KRSB=6707
        7002  BSW=7002
        3602  LOC=3602

        4000  *4000

US
04000  1000  START,  TAD MR0
04001  1206  CRCCHK, TAD L260
04002  6704          KLSA
04003  6706          KGOA
04004  6703          KSBF
04005  5204  RDCOD,  JMP .-1
04006  7264  L260,   CML STA RAL
```

C continued

```
US
04007  0000          KSEN                    S
04010  7610          SKP CLA
04011  3211          DCA .
04012  3636          DCA I PTR
04013  1205          TAD RDCOD
04014  6704          KLSA                    /LOAD INTO STA. REG. A
04015  6706  LOOP,   KGOA
04016  6701          KSDR
04017  5216          JMP .-1                 /WAIT FOR DATA FLAG
04020  7002          BSW
04021  7430          SZL
04022  1636          TAD I PTR
04023  7022          CML BSW
04024  3636          DCA I PTR
04025  7420          SNL
04026  2236          ISZ PTR
04027  2235          ISZ KNT
04030  5215          JMP LOOP
04031  7346          STA CLL RTL
04032  7002          BSW
04033  3235          DCA KNT
04034  5201          JMP CRCCHK
04035  7737  KNT,    7737
04036  3557  PTR,    LOC-23
04037  7730  M50,    -50
                     $
```

```
KCLR=6700                                PALC-V1  01/04/73  PAGE 1-1

CRCCHK  4001
KNT     4035
LOC     3602
LOOP    4015
L260    4006
M50     4037
PTR     4036
RDCOD   4005
START   4000
```

D

```
^C
^C
```

E

```
.R EDIT

*INPUT FILE-2:TEST.PAL
*OUTPUT FILE-1:TEST.PAL
```

```
F   #R

    #$,'L
    START,  TAD MR0

    #.S
    START,  TAD MR\50

G   #. 7L
                KDEN

    #.S
                KD\SEN

    #.L
                KSEN

H   #. 5S
                KLSA                /LOAD INTO STA. REG. A

I   #$1"L
                JMP .-1 /WAIT FOR DATA FLAG

    #.S
                JMP .-1             /WAIT FOR DATA FLAG

J   #E
```

```
K   .R PALC/N

    -INPUT FILES
    *1:TEST.PAL
    *
    -BINARY FILE
    *2:TEST.BIN
    -LIST TO
    *TTY

    BINARY FILE ON UNIT 2

    BINARY FILE CLOSED
    CRCCHK 4001
    KNT    4035
    LOC    3602
    LOOP   4015
    L260   4006
    M50    4037
    PTR    4036
    RDCOD  4005
    START  4000
```

```
L   ↑C
    ↑C
    .DEL 2:TEST.PAL

    .
```

A The Keyboard Monitor is loaded and the DAte command is used to indicate the current date. The user requests a directory listing of cassette drive 1; he decides to zero the directory, thereby deleting all files present on the cassette.

B PALC is called from the System Cassette. The input file, TEST.PAL, is stored on cassette drive 2. The user decides to specify as output only a listing on the line printer. Two errors are flagged by the assembler and printed on the console terminal during its second pass.

C The listing is printed on the line printer. The user then marks this listing with appropriate corrections and insertions.

D PALC prints a ↑C; the user makes sure that the System Cassette is still mounted on drive 0 and then types a ↑C to cause control to return to the Monitor.

E The Editor is next called from the System Cassette so that the errors in the file TEST.PAL may be corrected. The input is again drive 2, and the output file will be sent to the previously zeroed cassette on drive 1.

F The R command brings in the first page of text and the intra-buffer search is used to find the first error. This misspelling is corrected using the single character search command and the rubout character.

G The next mistake occurs 7 lines further in the listing; the incorrect character is found and corrected. The line is also listed to make sure the correction was made properly.

H The user inserts a comment in the 5th line forward from this line by skipping ahead 5 lines, searching for the A, and then adding the comment to the line.

I The intra-buffer search is used to locate the next correction; a tab is inserted between the 1 and the tab already present.

J The file is closed.

K The user calls PALC again, specifying the edited file on drive 1 as the input. The /N option is used to obtain only a listing of the symbol table; the binary file is output to drive 2, and the symbol table is listed on the console terminal.

L After assembly, PALC prints ↑C and waits while the user makes sure the System Cassette is mounted on drive 0. He then types ↑C and control returns to the Monitor. The user deletes the first input file (the uncorrected TEST.PAL) from the cassette on drive 2 to complete his session.

APPENDIX E
MONITOR SERVICES

Included in this Appendix is information the user needs if he intends to create files using the PAL machine language or reference system device handlers.

E.1 MONITOR MEMORY MAP

The CAPS-8 Keyboard Monitor occupies the following memory locations; if the user's program does not overwrite these areas of memory, the routines they contain will be available for use from within his program and the Monitor may be restarted after execution. (Section E.2 provides more information concerning these routines.)

Table E-1 Monitor Memory Map

Address	Contents
	FIELD 0
7400	LPT and Console Terminal Handlers
7600	Bootstrap, KBD Handler, Interrupt Routine
	FIELD 1
5200-6200	Keyboard Monitor and Commands
6200	WAIT and part of Cassette HANDLER
6400	CLOSE and ENTER
6600	Cassette HANDLER
7000	LOOKUP
7200	UTIL and part of Cassette HANDLER
7400	Binary Loader
7600	Buffer

E.2 MONITOR SERVICE UTILITY SUBROUTINES

The user may direct his program to one of the following utility subroutines providing the routine has not been overwritten or otherwise destroyed.

Table E-2 Utility Subroutines and Locations

Address Name	Location	Service
LPOCHR	07400	This routine is used to print a character on the line printer. The calling sequence is:

Table E-2 Utility Subroutines and Locations (Cont'd)

Address Name	Location	Service
		CDF (current field) CIF 0 TAD character JMS I (LPOCHR The character in bits 5-11 of the AC is added to the line printer ring buffer to be printed.
TTOCHR	07402	This routine is used to print a character on the console terminal. The calling sequence is: CDF (current field) CIF 0 TAD character JMS I (TTOCHR The character in bits 5-11 of the AC is added to the teleprinter ring buffer to be printed. (The character will not print if ECHO is off at the time, but can be designated as a "must print" character by turning AC bit 3 on. This causes the character to force ECHO on.)
LPPUTP	07404	This address contains the next free location in the line printer buffer.
LPGETP	07405	LPGETP contains the previous location which was output in line printer buffer (never a pointer).
LPCHCT	07406	This location contains the number of line printer interrupts yet to be expected.
ECHO	07407	If this address contains -1, ECHO is off (no ECHO); if it is set to 0, ECHO is on.
TTSIZ	07410	This address contains the length of the teleprinter ring buffer (number of characters it can hold).
TTPUTP	07411	TTPUTP contains the next free location in the teleprinter output buffer.
TTGETP	07412	This address contains the last location which was output in the teleprinter buffer (never a pointer).

Table E-2 Utility Subroutines and Locations (Cont'd)

Address Name	Location	Service
TTCHCT	07413	TTCHCT contains the number of teleprinter interrupts yet to be expected.
LPSIZ	07414	This location contains one less than the length of the line printer ring buffer (number of characters it can hold -1).
MONRES	07415	MONRES is the location in field 0 which can be branched to in order to restart the Keyboard Monitor in memory (assuming the Keyboard Monitor has not been destroyed). Control jumps from this location to the routine MON.
	07600	Branching to this location causes a complete rebootstrap of the Keyboard Monitor from the System Cassette on drive 0. If an I/O error occurs or if the cassette on drive 0 does not contain the file MONTOR.BIN, the system waits for the user to mount a good System Cassette; typing a CTRL/C will then rebootstrap.
KBDFLG	07601	If this location contains a non-zero number, it signifies that a character (other than CTRL/O or BREAK) was typed on the keyboard and has not yet been read. It is lost if a second character gets typed before the previous one is read.
KBDIN	07602	This location contains the last character typed on the keyboard. (Here BREAK and CTRL/O do count.)
BREAK	07603	This location contains a 1 if a BREAK has not been used; 0 if it has.
CTRLCJ	07604	If this location is 0, then whenever CTRL/C is typed, the Monitor will branch to 07600 and bootstrap (after the current cassette operation finishes). If not 0, then when the current cassette operation finishes, control is transferred (with interrupts on) to this location in field 0. Set this to point to MONRES if the Keyboard Monitor has not been destroyed.

Table E-2 Utility subroutines and Locations (Cont'd)

Address Name	Location	Service
———	07605	Same as 07600.
KBDCHR	07626	This routine reads a character from the keyboard. It waits for KBDFLG to be non-zero, then zeroes it and returns the contents of KBDIN in the AC. The calling sequence is: CDF (current) CIF 0 JMS I (KBDCHR
DISMIS	07645	The system branches to this location to dismiss an interrupt.
INTRPT	07657	The interrupt routine begins at this address.
LPBUFR	07731	LPBUFR is the start of the default line printer ring buffer (initially of length 2).
TTBUFR	07734	TTBUFR is the start of the default teleprinter ring buffer (initially of length 30).
MONSTART	15200	Branching to this location starts the Monitor, assuming that the entire Monitor is still in memory. The routine waits for TTY and LPT then resets buffers to defaults and empties them. Sets CTRLCT to point to MONRES. Sets ECHO on and sets BREAK to 1. Notes cassettes as not being in use.
MON	15201	This routine restarts the Monitor but does not do any of above.
	15400	Starting at this location also restarts the Monitor and resets locations that may be in a temporary state if the Monitor has been stopped (e.g., by hitting STOP) prematurely.
WAIT	16200	This routine waits for the last cassette operation (if there was one) to complete. The calling sequence is:

Table E-2 Utility Subroutines and Locations (Cont'd)

Address Name	Location	Service
		CDF (current) CIF 10 JMS I (WAIT <error return> <normal return> If an error return is taken, bit 0 may be on and bits 4-11 will contain the contents of status register B at the time of error. This routine should be called sometime after every call to HANDLER.
CINUSE	16273	If this location contains a 0, cassettes are ready; 1 means cassettes are in use; -1 means cassettes had an error in a previous operation.
BSTATE	16274	This location contains the status of register B at termination of cassette operation.
CLOSE	16400	Calling this subroutine terminates an output file and writes a new sentinel file at the end of the cassette. The calling sequence is: TAD (UNIT CDF (current field) CIF 10 JMS I (CLOSE <error return> <normal return> The error return is taken only if an end-of-tape is encountered before the sentinel file is successfully written.
BACK	16402	This routine positions the cassette so that the header record of the current file may be written over. The calling sequence (field 1 only) is: JMS I (BACK <error return> <normal return>
ENTER	16404	Calling this subroutine opens a new file on a cassette. The calling sequence is:

Table E-2 Utility Subroutines and Locations (Cont'd)

Address Name	Location	Service
		TAD (UNIT CDF (current field) CIF 10 JMS I (ENTER <error return> <normal return> where UNIT is the cassette unit drive number. Before making this call, the user must set up the new filename in an in-core header record known as the SINCH. ENTER automatically puts the date and record size into the SINCH for the user. (The SINCH format is described in Section E.4). The INCH is destroyed.
HANDLER	16600	This routine calls the system cassette handler which is resident and will be used by all system programs. The handler routine is also available to any user who does not load over it. Before calling this handler, the user must ensure that the cassette is correctly positioned. See also LOOKUP, ENTER, and CLOSE. The calling sequence is: CDF (current field) CIF 10 TAD (UNIT JMS I (HANDLER ARG1 (function control word) ARG2 (buffer address) <error return> <normal return> The unit number is left in the AC. Only bits 8-11 are used (units 0-17 octal). However, to specify unit 0, at least one other bit (of bits 0-7) must be on. It is more convenient, therefore, to leave the unit number as a character in the AC (0-9 would be 60-71). A real 12-bit 0 in the AC means use the previous unit. (The initial unit is 0.) The function control word has the following form: Bit 0: 0 means read 1 means write Bits 6-8: field of buffer Other bits are ignored.

Table E-2 Utility Subroutines and Lcoations (Cont'd)

Address Name	Location	Service
		The length of the record (in 8-bit bytes) must have been previously stored in location BSIZE. The specified record size must be between 1 and 377. LOOKUP and ENTER return with BISZE set to 200 (octal), the usual record size. If an error return is made, the AC bits 4-11 specify the contents of status register B when the error occurred. These bits are summarized below: bit 4: CRC/block error bit 5: timing error bit 6: EOT/BOT bit 7: EOF bit 8: drive empty bit 9: rewind bit 10: write lock out bit 11: ready Bit 0 of the AC will be a 1 if the error occurred on the previous handler call as opposed to the current handler call. This is because the handler will wait (by calling WAIT) if it is called while it is already in use. The user can manually wait for the cassette operation to be completed by calling WAIT. If an error occurs, bit 0 of the AC will always be on in this case. The user may check to see if the handler is in use without waiting by interrogating the location CINUSE. Non-zero means that the handler is in use. Two successive calls to HANDLER should not be made without an intervening call to WAIT.
BSIZE	17000	This location contains the current record size.
LOOKUP	17002	Calling this subroutine positions a cassette at a specified file to be read. The calling sequence is:

Table E-2 Utility Subroutines and Locations (Cont'd)

Address Name	Location	Service
		TAD (UNIT CDF FROMFLD CIF 10 JMS I (LOOKUP CDF (filenamefld) ptr to filename <error return> <not found return> <found return> UNIT represents the cassette unit drive number. The header is put in the INCH. The filename consists of 11 consecutive ASCII characters.
UTIL	17200	This routine allows the user to specify that a utility operation be performed. The user must be familiar with the hardware specifications as described in the TU60 CASSETTE TAPE TRANSPORT MAINTENANCE MANUAL (DEC-00-TU60-DA) to understand what these operations do and what conditions cause errors. The calling sequence is: CDF n CIF 10 TAD (UNIT JMS I (UTIL utility code <error return> <normal return> The following are legal utility codes; all other codes are illegal. 10 rewind 30 backspace file gap 40 write file gap 50 backspace block gap 70 skip to file gap
OPT1 OPT2 OPT3	17400 17401 17402	Switch option characters (e.g., /A) stored as 36 bits for A-Z, 0-9 as shown in diagram in Figure E-1.

Table E-2 Utility Subroutines and Locations (Cont'd)

Address Name	Location	Service

OPT 1	A	B	C	D	E	F	G	H	I	J	K	L
OPT 2	M	N	O	P	Q	R	S	T	U	V	W	X
OPT 3	Y	Z	0	1	2	3	4	5	6	7	8	9

Figure E-1 Switch Option Characters

SINCH	17403	See ENTER.
DATE	17531-17540	These locations contain 8 characters representing the date (e.g., 01/22/73).
INCH	17600	See LOOKUP.

E.3 RING BUFFERS

Ring buffers must be located in upper core (4000-7777) of field 0. They consist of one or more buffer segments, each one of which consists of two or more consecutive locations (the last one pointing to the next segment). The last segment points to the first one. Ring buffers can be changed by System Programs.

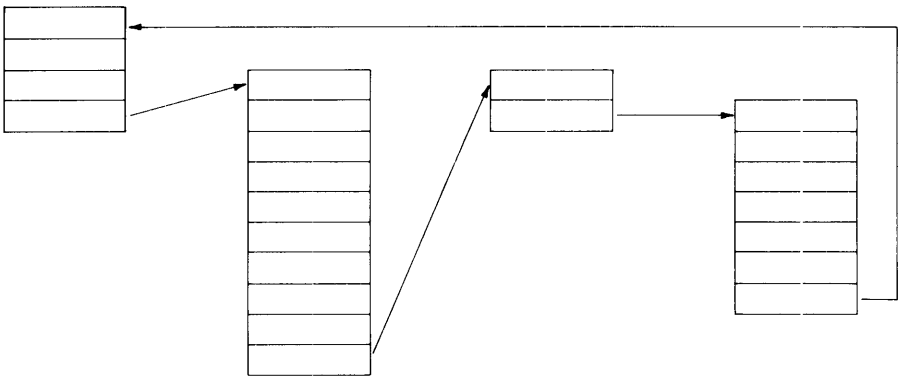


Figure E-2 Ring Buffers

E.3.1 Modifying the Ring Buffers

The initial ring buffer supplied by the Monitor consists of one buffer segment of length LPSIZE (TTSIZE) not counting the pointer. The first location is called LPBUFR (TTBUFR) and the last location is called LPBFND (TTBFND). The value LPSIZE-1 (TTSIZE) is stored in the location LPSIZ (TTSIZ). The buffer is initially empty. The next location free in the buffer is pointed to by LPPUTP (TTPUTP) and the previous location which has already been output is known as LPGETP (TTGETP). Both these locations point only to positive words, never to negative pointers. LPCHCT (TTCHCT) is the ones complement of the number of characters left in the buffer to be output if I/O is still in progress. (Specifically, it is the number of flags which have yet to come up.) LPCHCT (TTCHCT) is zero (0) if there is no output in progress.

To enlarge the ring buffer, wait until LPCHCT (TTCHCT) is zero. Then set LPBFND (TTBFND) to point to the start of the buffer and have the end of the buffer point to LPBUFR (TTBUFR). Change LPSIZ (TTSIZ) to be the length of the buffer (length -1 in case of LPSIZ) not counting pointer words. Interrupts may be on while this is done providing no LPT (TTY) I/O is initiated.

E.4 HEADER RECORD FILE STRUCTURE

All files the user creates must begin with a header record (40 octal bytes long), followed by 200 octal byte long records. The structure of a header record is as follows:

Table E-3 Header Record Structure

Bytes (octal)	Description
1-6	Filename; may consist of any alphabetic character or digit and is padded with spaces on the right.
7-11	Filename extension; see Table 2-1 for recommended extension names.
12	File type; maintained by the system for its convenience and for standard compatibility. File types are: 1 ASCII file 2 Standard DEC Binary File 12 Bad file (Specified for all deleted files) Refer to Section 2.2.1 for an explanation of these file types.
13-14	File record length; always has the value 0,200 (i.e., 200) for compatability with standards.
15	File sequence number (not used).

Table E-3 Header Record Structure (Cont'd)

Bytes (octal)	Description
16	Header continuation byte; always 0.
17-24	ASCII date stored as dd mm yy, or 6 spaces if no date was specified when the file was saved. This is the creation date of the file.
25	File version number. New files are version 0 and are automatically incremented by the CAPS-8 EDITOR.
26-40	Not used.

E.5 CAPS-8 BOOTSTRAPS

The CAPS-8 Hardware Bootstrap is used to load the Cassette Keyboard Monitor into memory. This bootstrap is stored in the computer in read-only-memory so that it is always available for use. Pressing the SW switch on the computer console causes this bootstrap to be executed; it calls the program C2BOOT.BIN into memory from the System Cassette. The CAPS-8 Hardware Bootstrap is comprised of the following instructions, included here for the user's information:

```
/CASSETTE SYSTEM BOOTSTRAP          PALC-V1  01/04/73  PAGE 1

                /CASSETTE SYSTEM BOOTSTRAP

                /       COPYRIGHT 1972
                /       DIGITAL EQUIPMENT CORPORATION
                /       MAYNARD, MASS.  01754

                /       S.K.

                /STARTING LOCATION (NORMALLY):  4000
                /STARTING LOCATION FOR OS/8:    5777

          6700          KCLR=6700
          6701          KSDR=6701
          6702          KSEN=6702
          6703          KSBF=6703
          6704          KLSA=6704
          6705          KSAF=6705
          6706          KGOA=6706
          6707          KRSB=6707
          7002          SSW=7002          /PDP-8/E, -8/F, AND -8/M ONLY
          5602          LOC=5602          /LOCATION WHERE SECONDARY
                                          /BOOTSTRAP REALLY GETS LOADED

                        IFDEF OS8          <*3777;CLL>
          4000          *4000             /INITIALIZE PULSE CLEARS THE LINK
04000     1237  START,  TAD M50           /CHANGE READ CRC CODE (6) TO
                                          /REWIND <1>       [BIN]
04001     1206  CRCCHK, TAD L260          /LOAD READ CRC CODE INTO STATUS
                                          /REGISTER A       [JMP I START]
04002     6704          KLSA              /FIRST TIME THROUGH, LINK MUST
                                          /BE 1 HERE
04003     6706          KGOA              /INITIATE THE OPERATION (READ
                                          /CRC OR REWIND OR FRWD FILE GAP)
04004     6703          KSBF              /READY?
04005     5204  RDCOD,  JMP .-1           /NO, WAIT
04006     7264  L260,   CML STA RAL       /SET L=1 AND AC= A HALT (7776)
04007     6702          KSEN              /ANY ERRORS?
04010     7610          SKP CLA           /NO
04011     3211          DCA .             /HALT ON ANY ERROR EXCEPT FOR
                                          /REWIND OR FRWD FILE GAP
04012     5636          DCA I PTR         /CAN'T ALLOW 'TAD I PTR' LATER
                                          /TO AFFECT LINK
04013     1205          TAD RDCOD         /GET CODE FOR READ (0)
04014     6704          KLSA              /LOAD INTO STATUS REGISTER A
```

```
04015  6706  LOOP,   KGOA            /FIRST TIME STORES 173 INTO MEMORY
                                     /(8-BIT COMPLIMENT OF RDCOD)
                                     /OTHER TIMES READS ONE 6-BIT
                                     /BYTE OF PAIR
04016  6701           KSDR            /NEW DATA WORD READY?

04017  5216           JMP .-1         /NO, WAIT
04020  7002           BSW             /MOVE 6-BIT BYTE TO H.O. AC
04021  7430           SZL             /WHICH 6-BIT BYTE OF THE PAIR?
04022  1636           TAD I PTR       /2ND. SO ADD IN 1ST BYTE
04023  7022           CML BSW         /SWAP BACK AGAIN. SET LINK TO
```



```
                                     /INDICATE NEXT BYTE
04024  3636           DCA I PTR       /STORE BACK INTO MEMORY
04025  7420           SNL             /ARE WE DONE LOADING BOTH 6-BIT
                                     /BYTES?
04026  2236           ISZ PTR         /YES, SO POINT TO NEXT MEMORY WORD
04027  2235           ISZ KNT         /BUMP COUNTER
04030  5215           JMP LOOP        /REITERATE
04031  7346           STA CLL RTL
04032  7002           BSW             /SET AC=7577
04033  3235           DCA KNT         /SET COUNT  TO ALLOW READING A
                                     /200 BYTE RECORD
04034  5201           JMP CRCCHK      /GO CHECK THE CRC
04035  7737  KNT,     7737            /ONES COMPLIMENT OF NUMBER OF
                                     /BYTES TO LOAD
04036  3557  PTR,     LOC-23          /MEMORY LOCATION TO BEGIN LOAD AT
04037  7730  M50,     -50             /CLA SPA SZL
               /THIS ROUTINE BINARY LOADS BINARY FILES INTO MEMORY.
               /IT BEGINS BY LOADING A RECORD OF SIZE 40,
               /THEN CONTINUES TO LOAD SUCCESSIVE RECORDS EACH OF SIZE
               /200.
               /THIS PROCESS CONTINUES UNTIL IT DESTROYS ITSELF.
               /(LOCATIONS 4000 AND 4001 ARE REPLACED BY JMP I(BIN)
               /BY THE SECONDARY BOOTSTRAP.
               /THE FIRST MEMORY LOCATION BEFORE A NEW CASSETTE RECORD
               /IS READ IN IS LOADED WITH A RANDOM VALUE (173).
               /SUCCESSIVE WORDS ARE LOADED WITH THE 12-BIT QUANTITY,
               /100A+B, WHERE A AND B ARE SUCCESSIVE 6-BIT BYTES FROM
               /THE CASSETTE RECORD.
               /MEANINGLESS WORDS GET LOADED IF THE CASSETTE CONTAINS
               /8-BIT BYTES AS CAN (AND DOES) HAPPEN WHEN 'LOADING'
               /THE HEADER, AND WHEN 'LOADING' THE ORIGIN AT THE
               /BEGINNING OF THE RECORD.
```

```
CRCCHK  4001
KNT     4035
LOC     3602
LOOP    4015
L260    4006
M50     4037
PTR     4036
RDCOD   4005
START   4000
```

C2BOOT.BIN is the bootstrap which loads the Keyboard Monitor into memory. It is stored on the System Cassette and is comprised of the following instructions:

```
/ SECONDARY BOOTSTRAP                PALC-V1              PAGE 1

                    / SECONDARY BOOTSTRAP
                    /       COPYRIGHT 1972
                    /       DIGITAL EQUIPMENT CORPORATION
                    /       MAYNARD, MASS.  01754

                    /       S.R.
                7002        BSW=7002
                6701        KSDR=6701
                6701        KSDF=6701
                6703        KSBF=6703
                6704        KLSA=6704
                6706        KGDA=6706
                6707        KRSB=6707
                            NOPUNCH
                3602        *3602
                            ENPUNCH

03602  7240  BIN,    STA
03603  3317           DCA GRANT
03604  1253  ITSFLD,  TAD CDF0
03605  3230           DCA FLD
03606  3305  ITSORG,  DCA ORG          /ASSUMES ORIGIN ALWAYS APPEARS
                                       /AFTER FIELD SETTING
03607  7240  BINLDR,  STA
03610  3307           DCA ORGSW
03611  4243           JMS GETBYT
03612  7002  POOL,    BSW
03613  7112           CLL RTR
03614  7430           SZL
03615  5235           JMP SPEC          /BIT 4=1
03616  7510           SPA               /BIT 4=0 (TWO WORD COMMAND)
03617  2307           ISZ ORGSW         /IS ORIGIN
03620  7000  N7000,   NOP
03621  7004           RAL
03622  7104           CLL RAL
03623  3306           DCA TEM
03624  4243           JMS GETBYT
03625  1306           TAD TEM           /COMBINE
03626  2307           ISZ ORGSW
03627  5206           JMP ITSORG        /ORIGIN
03630  7402  FLD,     HLT
03631  3705           DCA I ORG
03632  2305           ISZ ORG
03633  6201  CDF0,    CDF 0
03634  5207           JMP BINLDR
03635  7500  SPEC,    SMA
03636  5302           JMP     MON
03637  7005           RTL
03640  0220           AND N7000
03641  7002           BSW
03642  5204           JMP ITSFLD
03643  0000  GETBYT,  0
03644  2317           ISZ     GRANT
03645  5274           JMP     RDBYTE
03646  1311           TAD     X260
```

```
03647  6774           KLSA
03650  6726           KGOA
03651  6725           KSDF
03652  5251           JMP     .-1
03653  6727           KRSB
03654  0026           AND     X7774
03655  7041           SZA CLA
03656  7402           HLT
03657  2521           ISZ     FIRST
03660  5265           JMP     NOTFST
03661  4363           JMS RESET
03662  1312           TAD     X270
03663  7121           STL
03664  5716           JMP I   X4002
03665  1322  NOTFST,  TAD     X7600
03666  3317           DCA     GRKNT
03667  1313           TAD     X200
03670  6774           KLSA
03671  6726           KGOA
03672  6771           KSDF
03673  5272           JMP     .-1
03674  6746  KOMYIE,  KGOA
03675  6731           KSDF
03676  5275           JMP     .-1
03677  5643           JMP I   GETBYT

03700  0000           /BLOCK 3701-.    /LOCATION 3701 IS SKIPPED BY
                                       /PRIMARY LOADER

                      NOPUNCH
03701  0000           0
                      ENPUNCH
03702  6215  KON,     CDF CIF 10
03703  5704           JMP I   .+1
03704  5200           5200

03705  0000  ORG,     0
03706  0000  TEM,     0
03707  0000  ORGSW,   0

03710  0200  X200,    200
03711  0260  X260,    260
03712  0270  X270,    270
03713  1737  X7737,   1737
03714  4035  XKNT,    4035
03715  4036  XPTR,    4036
03716  4002  X4002,   4002
03717  0000  GRKNT,   0
03720  7774  X7774,   7774
03721  7777  FIRST,   -1
03722  7600  X7600,   7600
03723  0000  RESET,   0                /SET UP PRIMARY BOOTSTRAP
                                       /FOR REUSE
03724  1313           TAD     X7737
03725  3714           DCA I   XKNT
```

```
03726  3715           DCA I   XPTR
03727  1352           TAD X3211       /A "DCA ." FOR LOCATION 4011
03730  3733           DCA I X4011
03731  5723           JMP I RESET
03732  3211  X3211,   3211
03733  4011  X4011,   4011
03734  0000           ZBLOCK 4000-.
04000  3602           BIN
04001  5600           JMP I .-1       /MUST END IN OCTAL 00
                      $
```



```
BIN     3602
BINLDR  3607
CDFM    3633
FIRST   3721
FLD     3634
FOOL    3612
GETBYT  3643
GRKM1   3717
ITSFLD  3604
ITSORG  3606
KSDF    6701
MQV     3722
NOTEST  3665
NZMPG   3624
ORG     3705
ORGSW   3707
RDBYTE  3674
RESET   3723
SPEC    3635
TEM     3706
YANT    3714
XPTR    3715
X2??    3710
X2??    3711
X2??    3712
X3211   3732
X4??2   3716
X4011   3733
X7???   3722
X7757   3715
X7774   3720
```

APPENDIX F
ASSEMBLY INSTRUCTIONS

CAPS-8 source programs are supplied on DECtape. These sources are assembled with PAL8 and copied to cassette with PIPC. To build the CAPS-8 system cassette with PIPC, the user must load the OS/8 cassette handlers as described in USING AND LOADING YOUR NEW OS/8 CASSETTE HANDLERS (DEC-S8-UCASA-A-D). The following instructions may be used to assemble the sources, print source listings, and create the CAPS-8 system cassette on drive 0.

```
.R PAL8
*C2BOOT,TEMP<C2BOOT
.R CREF
*TEMP
.R PAL8
*MONTOR,TEMP<CASMON
.R CREF
*TEMP
.R PAL8
*SYSCOP,TEMP<SYSCOP
.R CREF
*TEMP
.R PAL8
*EDITC,TEMP<EDITC
.R CREF
*TEMP
.R PAL8
*PALC,TEMP<PALC
.R CREF
*TEMP
.R PAL8
*CBASIC,TEMP<CBASIC/K
.R PIP
*LPT:<TEMP.LS

.R PIPC
*CSA0:</Z
*CSA0:C2BOOT<C2BOOT/B
*CSA0:MONTOR<MONTOR/B
*CSA0:SYSCOP<SYSCOP/B
*CSA0:EDITC<EDITC/B
*CSA0:PALC<PALC/B
*CSA0:CBASIC<CBASIC/B
*TTY:<CSA0:/L
```

Cassette Programming System Users Manual
DEC-8E-OCASA-B-D

READER'S COMMENTS

NOTE: This form is for document comments only. Problems with software should be reported on a Software Problem Report (SPR) form (see the HOW TO OBTAIN SOFTWARE INFORMATION page).

Did you find errors in this manual? If so, specify by page.

_____

_____

_____

_____

_____

_____

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

_____

_____

_____

_____

_____

_____

Is there sufficient documentation on associated system programs required for use of the software described in this manual? If not, what material is missing and where should it be placed?

_____

_____

_____

_____

_____

_____

Please indicate the type of user/reader that you most nearly represent.

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Non-programmer interested in computer concepts and capabilities

Name _____ Date _____

Organization _____

Street _____

City _____ State _____ Zip Code or Country _____

If you do not require a written reply, please check here. ☐

---------- Fold Here ----------

---------- Do Not Tear - Fold Here and Staple ----------

FIRST CLASS
PERMIT NO. 33
MAYNARD, MASS.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

Postage will be paid by:

digital

Software Communications
P. O. Box F
Maynard, Massachusetts 01754

HOW TO OBTAIN SOFTWARE INFORMATION

SOFTWARE NEWSLETTERS, MAILING LIST

The Software Communications Group, located at corporate headquarters in Maynard, publishes newsletters and Software Performance Summaries (SPS) for the various Digital products. Newsletters are published monthly, and contain announcements of new and revised software, programming notes, software problems and solutions, and documentation corrections. Software Performance Summaries are a collection of existing problems and solutions for a given software system, and are published periodically. For information on the distribution of these documents and how to get on the software newsletter mailing list, write to:

Software Communications
P. O. Box F
Maynard, Massachusetts 01754

SOFTWARE PROBLEMS

Questions or problems relating to Digital's software should be reported to a Software Support Specialist. A specialist is located in each Digital Sales Office in the United States. In Europe, software problem reporting centers are in the following cities.

Reading, England	Milan, Italy
Paris, France	Solna, Sweden
The Hague, Holland	Geneva, Switzerland
Tel Aviv, Israel	Munich, West Germany

Software Problem Report (SPR) forms are available from the specialists or from the Software Distribution Centers cited below.

PROGRAMS AND MANUALS

Software and manuals should be ordered by title and order number. In the United States, send orders to the nearest distribution center.

Digital Equipment Corporation
Software Distribution Center
146 Main Street
Maynard, Massachusetts 01754

Digital Equipment Corporation
Software Distribution Center
1400 Terra Bella
Mountain View, California 94043

Outside of the United States, orders should be directed to the nearest Digital Field Sales Office or representative.

USERS SOCIETY

DECUS, Digital Equipment Computer Users Society, maintains a user exchange center for user-written programs and technical application information. A catalog of existing programs is available. The society publishes a periodical, DECUSCOPE, and holds technical seminars in the United States, Canada, Europe, and Australia. For information on the society and membership application forms, write to:

DECUS
Digital Equipment Corporation
146 Main Street
Maynard, Massachusetts 01754

DECUS
Digital Equipment, S.A.
81 Route de l'Aire
1211 Geneva 26
Switzerland