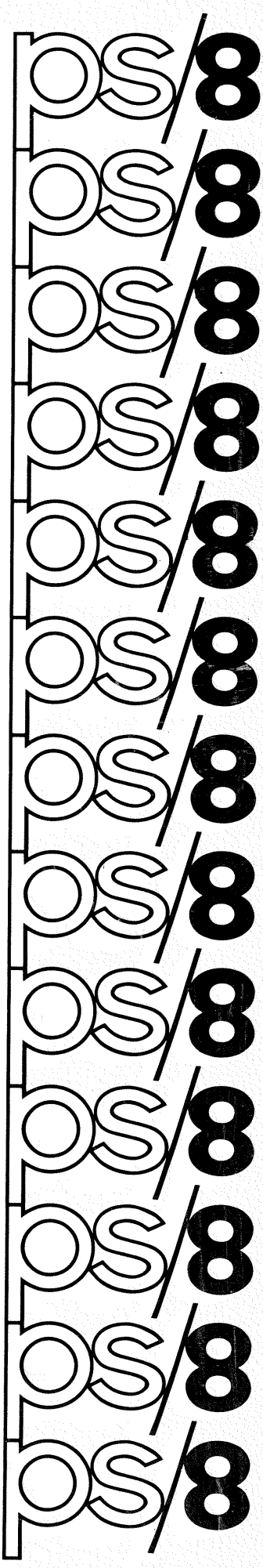
digital

8k programming
system user's guide



digital equipment corporation

DEC-08-MEFA-D

8K Programming

System User's Guide

This manual supersedes the information in Chapter 9 of Introduction to Programming 1970, and will replace that chapter in the next printing.

For additional copies, order No. DEC-08-MEFA-D from the Program Library, Digital Equipment Corporation, Maynard, Mass. 01754.

Price: $3.00

First printing, October 1970
Reprinted, February 1971

The PS/8 Software Support Manual
(DEC-P8-MEXB-D) will supersede the
PS/8 Programmer's Reference Manual
as of February, 1971.

PREFACE

This manual, the PS/8 System User's Guide, obsoletes the information in Chapter 9 of Introduction to Programming 1970, and describes the PS/8 Programming System for use by the average system user. Those programmers requiring a deeper insight into the system are advised to read the PS/8 Software Support Manual.

Two of the major components of PS/8, the Keyboard Monitor and Command Decoder, are described in Chapters 1 and 2, respectively. Each of the System library programs is described in detail as to its application to the PS/8 system. Instructions on using these library programs (for the benefit of beginning programmers) is contained in Introduction to Programming 1970 and Programming Languages.

The new 8K FORTRAN system is introduced in this document. The FORTRAN Compiler contains such features as implied DO loops, device-independent I/O, chaining, and other improvements. Detailed specifications on the 8K FORTRAN system appear in this volume.

The Appendices contain summaries of the ASCII and punched character sets, error messages, and permanent symbol tables. A demonstration program is included in Appendix D as an aid to understanding the abilities of PS/8.

CONTENTS

CHAPTER 6 UTILITY PROGRAMS

CHAPTER 7 THE 8K FORTRAN SYSTEM

CHAPTER 1

INTRODUCTION

The PS/8 Programming System is a program-development system, expandable to accommodate any amount of core memory from 8K up, and a wide range of input/output devices.

In addition to the PS/8 system executive routines, the system incorporates a great deal of additional software, including:

1. A Symbolic Editor (EDIT) used to create or modify source files for use by other system programs such as PAL-8, 8K SABR, and 8K FORTRAN.

2. PAL-8, an improved version of the 4K and 8K PAL-D assemblers, accepts source files written in assembly language and generates absolute binary code.

3. PIP, a completely new Peripheral Interchange Program, used to transfer files between devices, merge files, list and zero directories, and eliminate "holes" in PS/8 directories.

4. CONVRT, DECtape Conversion Program, a new program which provides ASCII file compatibility between the Disk/DECtape Monitor and TSS/8 format DECtapes and PS/8 files.

5. ABSLDR, Absolute Binary Loader, loads absolute binary files from devices into core.

6. ODT, Octal Debugging Technique, a new debugging package which includes all the features of the old ODT, but takes up none of the user's core space.

7. An improved version of the 8K FORTRAN language, consisting of the FORTRAN Compiler, SABR Assembler, a new Linking Loader, and the FORTRAN Library. The advantages of this new FORTRAN include:

a. Reducing the difficulty of using FORTRAN. If desired, a program can be compiled, loaded, and executed automatically by a single command.

b. Implied DO loops are permitted in the new PS/8 FORTRAN.

c. Program chaining greatly increases the power of the language.

d. Device-independent I/O has been added.

PS/8 provides true device-independence. For the first time on a PDP-8 computer, programs can be written without concern for specific I/O devices. In running a program the user can select the most effective I/O devices available. Further, if the system configuration is altered, programs need not be rewritten to take advantage of the new configuration.

The PS/8 system controls the copying of data from any medium to any other medium by means of subroutine calls to executive I/O routines. Logical names can be assigned to devices within the system to enable symbolic referencing of devices.

Variable length I/O buffers can be specified by the user program. Large buffers ensure efficient use of storage devices and a minimum of time spent in data transfer operations by minimizing disk and tape motion. PS/8 takes full advantage of the RK8 disk pack for fast bulk storage, yet full system services are possible with a single DECtape.

The discussion of the PS/8 Programming System in this manual assumes that the reader is familiar with 8K PAL-D, 8K SABR, 8K FORTRAN, and the Symbolic Editor as described in Programming Languages and Introduction to Programming 1970.

The reader need not be familiar with monitor systems, the Peripheral Interchange Program (PIP), ODT, the Absolute Binary Loader, the Linking Loader, CONVRT (the program to convert 4K Disk Monitor and TSS/8 ASCII files to PS/8 compatible files), or the fine details of the languages as these are explained in this document.

1.1 HARDWARE CONFIGURATIONS

The PS/8 Programming System can operate using either disk or DECtape as the system device. To accommodate PS/8 the disk configuration should have 64K or more words of storage and either a DECtape or high-speed paper tape reader/punch (to load the system and handle I/O).

The minimum PS/ configuration is a PDP-8/I, 8/L, 8/E, or PDP-12 with 8K of core, one DECtape used as the system device, and a 33-ASR Teletype terminal. A multiple DECtape system performs appreciably faster than a single DECtape system. The multiple DECtape system reduces DECtape motion because it is possible to copy directly (without intermediate searching) from the system DECtape to another DECtape (or vice versa) when editing or assembling.

A typical medium-sized system might contain a PDP-8/I, 8/L, 8/E, or PDP-12 with at least 8K of core, a high-speed paper tape reader/punch, and an RK8 disk pack and control. A disk system offers the additional convenience of easy and fast access to files, and large amounts of storage.

Up to fifteen devices can be interfaced to a single PS/8 system. These optional devices include:

up to 8 DECtape units (TC01/TU55 or TC08/TU56)
high-speed paper tape reader/punch
up to four RK8 disks
up to four RF08 disks
up to four DF32 disks
card reader
line printer

PDP-12 LINCtape

any other device for which it is possible to write a device handler in one or two pages of core.

1.2 SYSTEM SOFTWARE COMPONENTS

The main software components of the PS/8 system are five:

Keyboard Monitor,
Command Decoder,
library of system programs,
device handlers, and
User Service Routine (USR).

The Keyboard Monitor accepts commands from the Teletype keyboard to create logical names for devices, to run system and user programs, to save programs, and to call ODT. The Keyboard Monitor provides communication between the user and the PS/8 executive routines.

The Command Decoder is used when the programmer activates a system library program. The Command Decoder accepts a command string from the keyboard indicating input/output files. Following the keyboard command to run a system library program, the Command Decoder prints a star (*) and accepts a command line containing the files to be used as input, file name and destination of output, etc. The Command Decoder communicates between the user and the system library programs.

The library of system programs, as mentioned earlier, contains the Peripheral Interchange Program (PIP), Symbolic Editor, PAL-8 (the PS/8 version of 8K PAL-D), an Absolute Binary Loader, CONVRT (the file conversion program), a new, improved 8K FORTRAN, 8K SABR, and Linking Loader. Other system library programs will be added as they become available.

The User Service Routine (USR) controls the directory operations for the PS/8 system. A program can use the USR by means of standard subroutine calls such as are used to activate device handler subroutines. Some of the functions performed by the USR are as follows: loading device handlers, searching file directories, creating and closing output files, calling the Command Decoder, and chaining of programs. The details on the operation and use of the USR are contained in the PS/8 Programmer's Reference Manual (DEC-08-MEXA-D). For normal PS/8 usage, the USR functions unseen by the user and is of no concern.

When PS/8 is operating, the Command Decoder, Keyboard Monitor, and USR are swapped into core from the system device as required, and, when their operation has been completed, the previous contents of core are restored.

The core-resident portion of PS/8 is extremely small (256 words!) and allows for a maximum use of core by user programs.

CHAPTER 2

KEYBOARD MONITOR

2.1 SYSTEM CONVENTIONS

The PS/8 Programming System has various system conventions which are quickly mastered by even the novice programmer. Naming conventions for devices and file extensions have been designed as simple mnemonics.

PS/8 uses the words: "word", "page", "record", and "block" as units of storage. In directory listings and elsewhere file lengths are referenced in terms of blocks (or records). The terms are defined as follows:

$$1 \text{ block} = 1 \text{ record} = 2 \text{ pages} = 256_{10} \text{ words}$$

Each word is composed of 12 bits. The internal structure of the PDP-8 words and pages is described in detail in Introduction to Programming 1970.

2.1.1 Permanent Device Names

Each device in the PS/8 system is referenced by means of a standard permanent device name. These names are used in all I/O designations and are listed below:

Permanent Name	I/O Device
SYS	System device (disk if the system has a large disk - RK8 or RFØ8 - otherwise DTAØ)
DTAn	DECtape n, where n is an integer in the range 0 to 7, inclusive.
DSK	The default storage device for all files. The assignment of DSK is specified at system generation time. Usually DSK is the disk on a single disk system or DTAØ on a DECtape system.
TTY	Teletype keyboard and printer.
PTP	High-speed paper tape punch.
PTR	High-speed paper tape reader (system prints an up arrow (↑) to which the user replies by typing any key, before accepting input).
CDR	Card Reader

Permanent Name	I/O Device
LPT	Line printer (performs a form feed before it begins printing output from a new program).

2.1.2 File Names and Extensions

Files are referenced symbolically by a name of up to six alphanumeric characters followed, optionally, by a period and an extension of two alphanumeric characters. The extension to a file name is generally used as an aid for remembering the format of a file.

In most cases, then, the user will want to conform to the standard file name extensions established for PS/8. To lessen the amount of typing required, the various system programs append assumed extensions to input and output files where specific extensions are not indicated.

If an extension is not specified for an output file, the system programs append assumed extensions. Where an extension is not specified for an input file, the system does a search for that file name with the logical default extension. Failing to find such a file, a search is then done for the original file without an extension. For example, if PROG were specified as an input file to PAL-8, the Command Decoder first looks for the file PROG.PA (since .PA is the standard extension for PAL-8 input files). If PROG.PA is not found, the Command Decoder tries to find the file PROG (with no extension).

TABLE 2.1

Assumed Extensions

Extension	Meaning
.SV	A core image file or SAVE file; appended to a file name by the R, RUN, SAVE, and GET keyboard Monitor commands.
.FT	An 8K FORTRAN source file.
.SB	An 8K SABR source file.
.PA	A PAL-8 source file.

Extension	Meaning
.BN	An absolute binary file, default extension for a Binary Loader input file. Also used as the default extension for PAL-8 binary output file.
.RL	A relocatable binary file, default extension for a Linking Loader output file. Also used as the default extension for an 8K SABR output file.
.MP	File containing a loading map (for the Linking Loader).
.LS	A PAL-8 or 8K SABR assembly listing output file.
.TM	Temporary file generated by FORTRAN or SABR for system use.

For example, if the user types:

```
.RUN DSK PROG
```

the file PROG.SV (on device DSK) is run if found. If the user types:

```
.RUN DSK PROG.A
```

then PROG.A is run, if found.

2.2 USE OF THE KEYBOARD MONITOR

While using the system library programs the user can alert the Keyboard Monitor by typing CTRL/C (hold down the CTRL key and type the C key), which echoes at the teleprinter as ↑C. The Keyboard Monitor signals that it is ready to accept input by printing a dot (.) at the left margin of the teleprinter paper.

Each command to the Keyboard Monitor is typed at the keyboard and corrected, if necessary, before entering the line to the system. A line is entered to the system by typing either the RETURN key (which causes a carriage return/line feed operation, but no printed character) or the ALT MODE key (which prints a $).

Correcting typing mistakes is simple. The RUBOUT key is used to delete the last character typed. Pressing this key initially causes a backslash (\) character to be printed followed by the

character which was deleted. Successive RUBOUT keys 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 backslash to be printed (enclosing the deleted characters within backslashes). For example:

```
User types:              RUN DSK (RUBOUT) (RUBOUT) (RUBOUT) DTA1 FILE
Teleprinter shows:       RUN DSK\KSD\DTA1 FILE
Keyboard Monitor sees:   RUN DTA1 FILE
```

If at any time an input line becomes so corrected that it is no longer intelligible to the user, he can verify the contents of the line by typing the LINE FEED key. This causes the entire input line to be echoed as the Keyboard Monitor would see it at that point. The line is not considered to be entered to the system, and the user can proceed to edit, delete, or enter that line at his discretion.

For example:

```
User types:     RUN DTA3\3\2  PRG\G\OG (LINE FEED key typed)
System echoes: RUN DTA2 PROG
```

To delete a command line completely before it is entered, the user types CTRL/U (produced by holding down the CTRL key and typing the U key), which echoes as ↑U. To the Keyboard Monitor, CTRL/C is the same as CTRL/U, and returns control to the Keyboard Monitor without accepting the current input line.

2.3 COMMANDS TO THE KEYBOARD MONITOR

Any of nine commands can be typed in reply to the dot printed at the left margin of the teleprinter paper. Execution of these commands occurs only after typing the RETURN or ALT MODE key. Only the first two characters of any command are significant (need be typed).

Error messages given by the Keyboard Monitor are indicated under the command which could generate that message and in the Error Summary in Appendix B. Following an error message the system returns control to the Keyboard Monitor and the command must be reentered.

The generalized Keyboard Monitor error messages are described below:

TABLE 2.2

Keyboard Monitor Error Messages

Message	Meaning
XXXX?	Where XXXX is not a legal command, for example, if the user typed HELLO the system would echo HELLO?
TOO FEW ARGS	An important argument has been omitted from a command. For example, RUN DSK would generate this message.
SYSTEM IO ERROR	An error occurred while doing I/O to the system device.
MONITOR ERROR 5 AT XXXXX	An error occurred while doing I/O to the system device. This error is normally the result of not WRITE enabling the system device.

2.3.1 ASSIGN Command

The ASSIGN command is of the form:

```
.ASSIGN dev udev
```

or

```
.AS dev udev
```

The ASSIGN command causes a new, user-defined device name (udev) to be considered equivalent to the permanent device name (dev). Only one user name can be associated with a single device at a time. For example:

```
.AS DTA1 IN
```

causes all future references to IN to refer to DECtape unit 1, although references can still be made to DTA1.

If a user-defined device name is not indicated, any existing user-defined name is removed and only the permanent device name is valid. For example:

```
.AS DTA1 IN
.AS DTA1
```

The above sequence changes the name of DECtape 1 to IN and then back to DTA1 again.

The user-defined name is composed of up to four alphanumeric characters the first of which must be alphabetic. The user-defined name takes precedence over the permanent name. Device-independent programs are easily possible since a change in the user name of a device by means of the ASSIGN command can change the operation of a routine without changing the code.

User-defined device names should be one or two characters long, because all one and two character device names are unique. Due to the fact that the device name is internally coded in only one word, the three and four character names may not be unique. A three or four character name can be tested for uniqueness by typing an ASSIGN command as follows:

```
.AS name
```

and if a

```
name NOT AVAILABLE
```

message results, the name is unique within the current system and is not in the system tables; therefore it can be used.

2.3.2 DEASSIGN Command

The DEASSIGN command is of the form:

```
.DEASSIGN
```

or

```
.DE
```

and causes all permanent device names to be restored, discarding all previous user-defined device names. For example:

```
.AS DTA1 IN
.DE
```

causes DECtape 1 to be assigned the name IN. The DEASSIGN command removes the name IN from the system tables. DTA1 can no longer be referenced as IN.

2.3.3 GET Command

The GET command is of the form:

```
.GET dev file.ext
```

or

```
.GE dev file.ext
```

The GET command loads core image files (.SV format, not ASCII or binary) into core from a device. The device (dev) is specified along with the file name (file) and an optional file name extension (.ext). The file is loaded into core with its core control block. The core control block is then moved to a special area on the system device.

The core control block of a core-image file is maintained on the system device and contains information about the file such as its starting address, areas of core occupied by the file, and a Job Status Word. The Job Status Word is saved and loaded in location 7746 of field Ø with the file to indicate what parts of core the file uses and how, as follows:

TABLE 2.3

Job Status Word

Bit Condition	Meaning
Bit 0 = 1	File does not load into locations 0-1777 in field 0, (0000-1777).
Bit 1 = 1	File does not load into locations 0-1777 in field 1, (10000-17777).
Bit 2 = 1	Program must be reloaded before it can be restarted because it modifies itself during execution.
Bits 3-9	Unused, and reserved for future expansion.
Bit 10 = 1	Locations 0-1777 in field 0 need not be saved when calling the Command Decoder overlays.
Bit 11 = 1	Locations 0-1777 in field 1 need not be saved when calling the USR.

A core control block is created for each core image file (one which has been dumped from core onto some device by the SAVE command) when the file is created by the Linking Loader, ABSLDR, or the SAVE command.

If a file name extension is not specified to the GET command, the extension .SV (for core image file) is added automatically to the file name. For example:

```
.GE DTA3 OH
```

attempts to fetch the file OH.SV from device DTA3.

Four error messages are associated with the GET command. These are described below:

Message	Meaning
name NOT FOUND	The file with the name given was not found on the specified device.
name NOT AVAILABLE	The device with the name given is not available for use at the moment (check the device in question), or the user tried to obtain input from an output only device (such as the high-speed paper tape punch).
BAD CORE IMAGE	The file requested was not a core image file (it could have been an ASCII or binary file).
USER ERROR Ø AT xxxx	An input error was detected while loading the program. xxxx is meaningless

2.3.4 SAVE Command

The SAVE command is of the form:

```
.SAVE dev file.ext a-b,c,...;s=n
```

or

```
.SA dev file.ext a-b,c,...;s=n
```

where:

a-b,c ,...	the addresses of the areas and locations in core to be saved. Locations a through b, location c, and any other specified locations. a, b, and c are 5 digit locations. (The first digit represents the field.) When a single location is indicated (c) the entire page on which c is located is saved.
;s	the starting address of the file.
=n	where n is a four digit octal number representing the contents of the Job Status Word (see section 2.3.3).

The program currently in core is saved on the device (dev) specified, with the file name indicated (file.ext). If an extension is not specified, the extension .SV is automatically added by the system. If the remaining arguments are not given, the information they convey is taken from the current core control block (see section 2.3.3).

Restrictions on the arguments for the SAVE command are as follows:

a. Each set of limits (a-b) must be in the same field and not cross field boundaries.

b. No two sets of limits can overlap (a-b,c-d must not overlap).

c. No area to be saved can contain page 7600 or 17600, since these are the locations of the system resident I/O routines and system tables.

If an error message is printed in response to a SAVE command, the program currently in core has not yet been saved. Possible error messages are:

Message	Meaning
BAD ARGS	The arguments to the SAVE command are not consistent and violate restriction (b) above.
ILLEGAL ARG.	The SAVE command was not expressed correctly, illegal syntax used.
SAVE ERROR	An I/O error has occurred while saving the program. The program remains intact in core.
MONITOR ERROR 2 AT xxxx	Attempt made to output to a write-locked device, usually DECtape.
name NOT AVAILABLE	The device with the name given is not in any system table.

An example SAVE command is shown below:

```
.SAVE DSK OHLA.SV 55,1Ø5ØØ-1Ø577;1Ø5Ø2
```

This statement saves the program in core on the disk as a file named OHLA.SV. The areas of core saved are locations Ø to 177 in field Ø (when a single core location is indicated, the entire page on which that location occurs is saved) and locations 4ØØ to 577 of field 1. The starting address of the program is 5Ø2 in field 1. The core control block is updated to contain this information and the old Job Status Word is taken intact from the original core control block.

Example 2:

```
.SAVE DSK OHLA.SV
```

The above statement causes the program in core to be saved on the disk under the name OHLA.SV where the areas of core to be saved are taken from the original core control block.

2.3.5 ODT Command

The ODT command is of the form:

```
.ODT
```

or

```
.OD
```

This command causes the system ODT to be loaded into core and started. ODT is a system overlay, and, as such, takes up none of the user's program area unless the breakpoint feature is used, in which case ODT uses locations 4, 5, and 6 of every field in which a breakpoint had been placed. When using ODT to debug programs, the user-defined device names cannot be used. Each I/O device must be called by its permanent device name.

ODT is described further in section 6.3.

2.3.6 RUN Command

The RUN command is of the form:

```
.RUN dev file.ext
```

or

```
.RU dev file.ext
```

The RUN command, like the SAVE command, handles only core image files. The file indicated (file.ext) on the device (dev) specifield is loaded into core and its core control block is moved to the system scratch area. The program is started at its starting address. The RUN command is equivalent to a GET and a START command.

If an extension to the file name is not specified, the extension .SV is automatically added to the file name. For example:

```
.RU DTA1 PROG
```

causes the file PROG.SV on DECtape 1 to be loaded and started.

Possible error messages when using the RUN command are described below:

Message	Meaning
name NOT FOUND	The file with the name given was not found on the device indicated, or the user tried to input from an output only device.
BAD CORE IMAGE	The file indicated is not a core image file.
USER ERROR Ø AT xxxx	An input error occurred while loading the file. xxxx is meaningless.
name NOT AVAILABLE	The device with the name given is not listed in any system table.

2.3.7 R Command

The R command is of the form:

```
.R file.ext
```

and is similar to

```
.RUN SYS file.ext
```

This command handles only core image files from the system device. The file is loaded and started. If the file name extension is not specified, the extension .SV is automatically added.

The R command differs from the RUN command in that a core control block is not written to the system device. In order to SAVE a program which does not have its core control block in the usual location on the system device, all the optional arguments of the SAVE command must be explicitly stated. System programs are most often called using the R command, since they need not be saved. To call a user program to be later updated and saved, use either the RUN or GET command.

Error messages for the R command are the same as for the RUN command.

2.3.8 START Command

The START command is of the form:

```
.START nnnnn
```

or

```
.ST nnnnn
```

The program currently in core is started at location nnnnn. If the argument nnnnn is omitted, the program is started at the starting address specified in the core control block. For example

```
.ST 10555
```

starts the program in core at location 555 in field 1.

```
.ST
```

starts the program at the starting address given in the core control block.

The START command clears certain areas of core: the device handler in core table and the Command Decoder output area.

The error message:

```
NO!!
```

occurs if the user attempts to start (with .ST) a program which cannot be started. The user must not restart any user program or system library program which modifies itself while in core (bit 2 of the Job Status Word is set, see section 2.3.3).

2.3.9 DATE Command

The DATE command is of the form:

```
.DATE mm/dd/yy
```

or

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

The DATE command sets up the date in the system for purposes of dating directory entries and listings, printing on program output, etc. For example:

```
.DA 3/13/70
```

which indicates that the date is March 13, 1970.

The error message:

```
BAD DATE
```

is printed if the date is not entered correctly (using slashes).

CHAPTER 3

COMMAND DECODER

Once a system program has been called via the Keyboard Monitor, that system program uses the Command Decoder to obtain a list of I/O files and devices. The Command Decoder prints a star (*) at the left margin to indicate it is ready to accept a command string.

3.1 COMMAND DECODER CONVENTIONS

The Command Decoder uses the same keys as the Keyboard Monitor for the purpose of correcting typing mistakes. The RUBOUT key deletes one character per RUBOUT. The CTRL/U combination deletes an entire line. CTRL/C returns the user to the Keyboard Monitor, and the LINE FEED key causes the entire line to be printed on the teleprinter paper as it appears in the Teletype input buffer.

The description of files, file names, extensions, devices, and device names in Chapter 2 define system conventions for the Command Decoder as well as for the Keyboard Monitor (see sections 2.1, 2.1.1, and 2.1.2 for details).

3.2 COMMAND DECODER INPUT STRING

The expected string for I/O specification takes the form:

```
(output files) < (input files)
```

There can be 0-3 output files and 0-9 input files, depending on the requirements of the system program. The particular I/O string used with each system library program is described in the section on that program (later in the manual).

For example:

```
*DTA1:XY1,LPT:<DSK:PROG
```

The PAL-8 assembler would use the first output file (DTA1:XY1) for the binary output of the assembly and the second output file (LPT:) for the listing. DSK:PROG is the input source file.

Multiple file specifications are separated by commas. If no output files are indicated, the left angle bracket can be omitted. For example:

```
*DSK:PROG
```

would cause the file PROG on device DSK to be accepted as an input file. While the left angle bracket (<) is the accepted divider character between output and input files, the backarrow (←) is also acceptable.

File specifications for I/O files take the following forms:

Form	Meaning
device:filename	The specified file and the specified device, for example: DTA3:FILE1
device:	The specified device used as a non-directory device. Usually the device is a non-directory device anyway, such as LPT: If a directory device is used, the device can be read, but not written; for example, referencing DSK: causes the whole disk to be read. DSK: is always the default output device.
filename	The specified file on an assumed device. For output files and the first input file, the device is assumed to be DSK:, for example: *NAME<DTA2:PROG would indicate DSK:NAME as an output file. For input files after the first, the device is assumed to be the device of the previous entry. For example: *DSK:PROG1<DTA1:FILE1,FILE2,FILE3 will cause the three input files to be taken from DECtape 1.
(null file)	A null file (the absence of an explicit file specification) has different meanings in context and is indicated by a comma not followed by a file designation. For output files a null file indicates that there is no output file for this position. For example: *,LPT<DTA1:QUEST,DTA2:STAR If the preceding were an input to PAL-8 the first output file (binary) would not be generated, but the listing would be put on the line printer. For input files, a null file indicates that the device of the most recent entry is to be used as a non-directory device: *DSK:A<PTR:,, This input string would allow three paper tapes to be read from the high-speed reader.

3.2.1 Examples of Command Strings

Some examples of command strings specifying I/O are shown below with appropriate explanations. The command string precedes its explanation.

Example 1:

```
*DSK:BINARY,LPT:<SOURCE
```

The file named SOURCE is the input file on DSK. The two output files are: BINARY on DSK, and a second file on the line printer (LPT). The PAL-8 assembler uses this format; however, the assembler also adds the extension .BN onto the file labeled BINARY. Thus, the disk file will be named BINARY.BN (see section 2.1.2).

Example 2:

```
*INPUT1,INPUT2,INPUT3,PTR:,
```

This is a string of input files with no output file. Notice that the left angle bracket is not necessary if there are only input files specified. This type of input might be given to one of the loaders (which do not require output files). Three files are taken from device DSK and then two are taken from the paper tape reader (PTR:,).

Example 3:

```
*DTA2:A,B<XYZ:C,D
```

The input files C and D are taken from device XYZ (which could be any device with the user-defined name XYZ). The output files are a file named A on DTA2 and a file named B on DSK.

Example 4:

```
*,LPT:<SRC
```

The one input file is named SRC and is on DSK. The two output files specified are one null file (no output file in that position) and a file to be sent to the line printer (LPT).

Example 5:

```
*PTR:,,DTA1:X
```

This is another all input file string. The first input file comes from the paper tape reader as does the second (PTR:,). The third input file is named X and is on DTA1.

Example 6:

```
*A<TTY:,
```

The first input file comes from the Teletype (generally the low-speed reader), as does the second, null, file (TTY:,). The single output file is named A and is stored on DSK.

3.3 INPUT/OUTPUT SPECIFICATION OPTIONS

In addition to the listing of the output and input files that the Command Decoder accepts, there are various options which can be indicated to the system program on the file specification line. These options are unique to the individual system program and are covered in detail in the sections describing the various programs.

The user can (and is advised to) skip reading this section on his first pass through the book. This section describes the format of the options as part of an input line to the Command Decoder. When the reader is aware of the particular options he will use, this section will serve as a useful guide to formats.

Options are either numbers or single alphanumeric option characters.

Numbers used as options are set off from the command line with the equal sign (=) or square brackets ([]).

The alphabetic option characters are set off from the I/O specifications by the slash (/) character for single character options, and parentheses for a string of single characters. The usage of the slash, parentheses, equal sign, and square brackets is explained below. These explanations will serve as references and format specifications once the reader has learned which options he will be needing.

The format for input to the Command Decoder, then, looks generally like

the following:

```
output file(s)  <  input file(s)  (options)
```

3.3.1 The Slash Construction

A single alphanumeric option character is preceded by a slash and can occur anywhere in the input line, even in the middle of a name, although the usual position is after the input files. For example:

```
*TTY:/L<DSK:AB
```

is equivalent to:

```
*TTY:<DSK:AB/L
```

The option specified is L (this is a PIP command to list the DSK directory beginning at file AB).

3.3.2 The Parentheses Construction

Any number of option characters can be grouped together inside parentheses. This construction is also valid anywhere in the input line. For example:

```
*OUT:X<IN:Y (AQZ)
```

is equivalent to:

```
*OUT:X<IN:Y /A/Q/Z
```

3.3.3 The Equal Sign Construction

An octal number up to seven digits long can be used as an option indicator when preceded by an equal sign. This construction can only occur once in a line and must be followed by a separator character (comma, left angle bracket, backarrow, ALT MODE key or RETURN key) or by other options and a separator character. For example:

```
*FILE1=1ØØ2 (AXQ),FILE2
```

Three separate options are indicated: A, X, and Q.

3.3.4 The Square Bracket Construction

The square bracket construction can only occur immediately after an output file name and consists of an open bracket, a decimal number between 1 and 255, and a close bracket. The square bracket construction is never necessary and is only used by the more sophisticated user to optimize file storage.

The open bracket ([) is produced by holding down the SHIFT key while typing a K (i.e., SHIFT/K). The close bracket (]) is produced by typing a SHIFT/M.

This construction is used to provide an upper limit on the number of blocks (256 words per block) to be contained in the output file in order to allow the system to optimize file storage. For example:

```
*BINARY[19],LISTING[2ØØ]<SOURCE/8
```

The output files are a file named BINARY on the disk (DSK) with a maximum length of 19 blocks and a file name LISTIN (only six characters are significant) on the device DSK with a maximum length of 200 blocks. The input file is SOURCE on device DSK and the option specified is 8.

3.4 NOTES ON DEVICE HANDLERS

The device handlers supplied with the PS/8 system have certain operating characteristics which the user should understand. Most of these are extremely simple and require no action on the part of the user. Some device handlers perform additional operations for the user when I/O is being performed on a given device.

The device handler for the high-speed paper tape reader, before reading a tape[1], prints an up arrow (↑) and waits for the user to type any single character at the keyboard. This gives the user time to check the reader to be sure the tape is loaded in the reader, and it facilitates reading multiple tapes. Characters are read from the paper tape and packed into an input buffer. The end of the paper tape or a full input buffer causes the buffer to be made available to the user program. Typing CTRL/C while the tape is moving causes a return to the Keyboard Monitor.

The high-speed paper tape punch unpacks characters from the output buffer

[1]With PAL-8, the user is required to load the source tape three times for the three passes of the assembler. Each pass generates an up arrow.

and punches them on paper tape. Typing CTRL/C causes a return to the Keyboard Monitor. The punch must be manually turned on before an attempt is made to output to that device.

The line printer performs a form feed operation before beginning an output task. The characters are unpacked from the output buffer and printed. A form feed is also produced following the completion of an output task. Typing CTRL/C while the line printer is printing causes a return to the Keyboard Monitor. A CTRL/Z found in the output buffer causes printing to terminate and a form feed to be performed.

The Teletype handler performs I/O transfers between the Teletype keyboard and an input buffer or between an output buffer and the teleprinter. A CTRL/O typed while output is being printed terminates printing of the current output buffer. A CTRL/C typed at any time during input or output causes a return to the Keyboard Monitor. Typing CTRL/Z as input terminates input and gives an END OF FILE indication to the calling program. The teleprinter echoes all typed input and performs a line feed operation after any typed carriage return. The Teletype handler should not be used to read binary tapes from the low-speed reader.

The device handler for the card reader reads cards in alphanumeric format from either a punched card reader or a mark-sense card reader. Card format can have up to 80 characters per card, and trailing blanks are deleted from each card. Blank cards cause a carriage return/line feed to be entered into the data stream. Typing CTRL/C while cards are being read terminates reading and returns control to the Keyboard Monitor. Typing CTRL/Z terminates further reading and acts as if an end-of-file card was read. An end-of-file card contains a ← character in column 1 (an 0-8-5 punch) with the remaining columns blank. Either CTRL/Z or the end-of-file card is necessary to terminate reading. Because the card reader handler is a 2-page routine (the only current 2-page handler), the following programs cannot use this handler: ABSLDR, LOADER, and SABR. It is not possible to RUN or GET a program from the card reader as these commands assume a directory device.

Any DECtape other than the system device (if it is a DECtape System) can be interrupted with a CTRL/C, and control returns to the Keyboard Monitor. DECtape unit Ø on a DECtape system must never be WRITE LOCKED.

The disk and SYS device handlers work completely automatically without any user intervention.

3.5 COMMAND DECODER ERROR MESSAGES

The following is a complete list of the error messages which can be generated by an erroneous command string input to the Command Decoder.

Message	Meaning
ILLEGAL SYNTAX	The command line was formatted incorrectly.
TOO MANY FILES	More than three output files or nine input files were specified.
name DOES NOT EXIST	The device with the name specified could not be found in the system tables.
name NOT FOUND	The file with the name specified does not exist on the device indicated.

CHAPTER 4

PS/8 Symbolic Editor

The new PS/8 Symbolic Editor is used to create and/or modify source files (ASCII files) for input to other system programs such as FORTRAN, SABR, and PAL-8. The Editor contains commands for creating, modifying, or deleting characters, lines, or logical pages of text.

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 program listing. (This is not the same as a core memory page.) Editor operates on one page of text at a time, allowing the user to relate his editing to the pages of his listing. The Editor reads a page of text from the input file into its internal buffer where the page becomes available for editing. The Editor is able to accept as many as nine input files from various devices. Several powerful commands are available for automatic handling of files through the text buffer.

The low-speed paper tape reader is not recommended as an input device for the Editor. Input buffering may cause a loss of certain characters from low-speed reader.

4.1 CALLING AND USING THE EDITOR

To call the Editor from the system device, type

```
.R EDIT
```

followed by the RETURN key, where the dot was printed by the Keyboard Monitor. The system responds by alerting the Command Decoder and printing a star (*) at the left margin. In answer to the star, the user types his output file designation (1 allowed), a left angle bracket, and his input file designations (0 to 9 allowed).

For example:

```
*DSK:ABC<PTR:,DSK:AA1
```

causes input from the high-speed paper tape reader and from a file named AA1 on DSK. The output file is named ABC and stored on DSK.

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

The Editor can use any device which operates in ASCII mode and has a device handler in the system. For example, the high and low-speed reader/punch, DECtape, disk, card reader, and line printer are each legal devices for the Editor. Editor only operates properly on ASCII files. No error message is given if non-ASCII files are input to the Editor, but the results of operations are garbled.

As many as nine input files to the Editor are permitted. If the number of input files is zero, commands which attempt to read from a device are disabled, and a question mark (?) appears when a read is attempted. Where there are no input files, the A command is allowed and input is accepted from the keyboard. Only one output file is permitted from the Editor.

4.2 I/O SPECIFICATION OPTIONS

In section 3.3, the format for I/O specification options was described. These options are specified as part of the I/O specification line input to the Command Decoder. The following are the valid options for the Editor. (After reading this section the reader is advised to turn to section 3.3 to review the various formats.)

TABLE 4.1

Editor I/O Options

Option	Meaning
/A	Do not return to the Keyboard Monitor upon completion of a file close operation (E or Q command). Return control to the Command Decoder to specify new files for editing. The Editor is left in core. Executing an E command returns with clear text buffer; Q command retains the last text buffer.
/B	Convert two or more spaces to a TAB when reading from input device.

TABLE 4.1 (Cont'd.)

Editor I/O Options

Option	Meaning
/D	Delete the old copy of the output file (if one exists) before storing the new output file on the device. If /D is not used, the new file is put onto the output device under a system-coded name; the old file is deleted only after the new file is closed by an E or Q command and the new file is given its correct name.

For example:

```
*DTA2:FILE<DTA1:ARG/D
```

This command deletes FILE on DTA2 (if such a file exists) before creating a new FILE on DTA2.

4.3 SPECIAL KEY COMMANDS TO THE EDITOR

The Editor can be considered as operating in several different modes. Command mode is indicated by the Editor printing a # at the left margin of the teleprinter paper. This indicates that the Editor is waiting for a command from the keyboard.

Text mode is the condition of the Editor when it is processing various editing and I/O commands.

The following commands are available to allow the user to transfer between modes.

Command	Mode in Which Used	Meaning
CTRL/C	Text mode and Command mode	Return to the Keyboard Monitor. The text buffer is retained and the Editor remains accessible to the user. In Text mode, text between the last carriage return and the ↑C is lost.

The START command can be used to restart the Editor as follows:

```
↑C
.START
*
```

The above command (START) recalls the Command Decoder to accept new I/O file designations. When the START command is given and the previous output file was not closed, that output file and the contents of the output buffer are deleted.

Command	Mode in Which Used	Meaning
CTRL/O	Text mode	Stops the listing, writing, or punching of text on the Teletype. Returns control to the command mode.
CTRL/FORM	Text mode	Returns the Editor to command mode.

Other special Editor characters are used to represent numbers or perform erasures, as described below:

Character	Example	Meaning
.	.+1 C .-7 L . L	The dot or period character is used as the current line counter character. The dot can be used alone or with + or - an integer, any place where a number can be used.
/	/-7 L /-5 I	The slash character is similar in use to the dot and represents the highest numbered line in the text buffer.
RUBOUT key	\	Typing the RUBOUT key in text mode causes a backslash to be printed and one character deleted from the text buffer. The erasure is done right to left up to the last carriage return/line feed. Typing the RUBOUT key in command mode causes the entire command line to be deleted. Typing the RUBOUT key inside a search string causes the string to be deleted and the $ reprinted.

4.4 SUMMARY OF EDITOR COMMANDS

The following is a summary of the Symbolic Editor commands. Additional information on these commands and their usage is found in Chapter 6 of Introduction to Programming 1970.

The commands discussed in this section can each be given whenever the Editor prints a # at the left margin. The commands are of the form:

```
#X      (RETURN key)
#n X    (RETURN key)
```

or

```
#m,n X  (RETURN key)
```

where m,n represents line number designations, n represents a single line number, and X represents the command letter. The command is entered to the Editor with the RETURN key. Numbers used in Editor commands are decimal numbers.

TABLE 4.2

Symbolic Editor Commands

Command	Format	Meaning
A	#A	Append the following text being typed at the keyboard until a FORM FEED (ASCII 214 or CTRL/FORM) is found. The FORM FEED returns control to command mode. Text input following the A command is appended to whatever is present in the text buffer.
L	#L	List entire contents of the text buffer on the teleprinter.
	#n L	List line n of the text buffer on the teleprinter.
	#m,n L	List lines m through n of the text buffer on the teleprinter. Control then returns to command mode.
LINE FEED key		Equivalent to .+1 L, lists the next line in the text buffer on the teleprinter.
>	#>	Equivalent to .+1 L, lists the next line in the text buffer on the teleprinter.
<	#<	Equivalent to .-1 L, lists the previous line in the text buffer on the teleprinter.
I	#I	Insert whatever text is typed before line 1 of the text buffer. The FORM FEED (CTRL/FORM) terminates the entering process and sends control to the command mode where Editor prints a #.
	#n I	Insert whatever text is typed (up to a FORM FEED) before line n of the text buffer.
C	#n C	Change the text of line n to the line(s) typed after the command is entered, up to a FORM FEED.
	#m,n C	Delete lines m through n and replace with the text line(s) typed after the command is entered. Typing CTRL/FORM indicates the end of the inserted lines.
D	#n D	Delete line n from the buffer.
	#m,n D	Delete lines m through n from the buffer. The space used by the deleted line(s) is not reused until after a K command is performed.
K	#K	Kill the buffer. Resets the text buffer pointers so it appears that there is no text in the buffer.

TABLE 4.2 (Cont'd.)

Symbolic Editor Commands

Command	Format	
M	#m,n$x M	Move lines m through n directly before line x in the text buffer. The $ character represents typing the dollar sign key (SHIFT/4). The old occurrence of the moved text is removed; no buffer space is lost.
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 one of the following: space (ASCII 240) / (ASCII 257) TAB (ASCII 211) RETURN (ASCII 215) At the termination of a G command, control goes to command mode with the current line counter equal to the line just listed.
	#n G	Get and list the first line which begins with a label, starting the search at line n.
B	#B	List the number of available core 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 accommodated in this area, multiply the number of free locations by 1.7.
S	#S	Character search command (see section 4.5).
J	#J	Inter-buffer search command for character strings (see section 4.5.2).
F	#F	Follows a J command only. Look for next occurrence of previously specified character string (see section 4.5.2).
$	#$ text' #$ text" #"	Performs a character string search for the string "text". See section 4.5.2 Following a string search, #" searches for the next occurrence of the string.
R	#R	Read from the previously specified input device and append the new text to the current contents of the buffer. If no input file was indicated or if no input remains, a ? is printed and control returns to command mode.
N	#N	Write the current buffer to the indicated output file and read the next logical page.
	#n N	Write the current buffer to the output file, zero the buffer, and read the next logical page. This is done n times until the nth logical page is in the text buffer. Control then returns to command mode.

TABLE 4.2 (Cont'd.)

Symbolic Editor Commands

Command	Format	Meaning
		The N command cannot be used with an empty text buffer. A ? is printed if this is tried.
Y	#n Y	Skip to a logical page in the input file, not writing any output. For example: #5 Y reads through four 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.
P	#P	Write the entire text buffer to the output buffer.
	#n P	Write line n of the text buffer to the output buffer.
	#m,n P	Write lines m through n, inclusive, to the output buffer. The lines m through n are written into the output buffer, and when this buffer is full, the text is output to the indicated output file.
T	#T	Punch trailer tape. Causes 32 frames of blank tape to be written into the output buffer (only to non-directory devices).
E	#E	Output the current buffer and transfer all input to the output file, closing the output file.
Q	#Q	Immediate end of file. Q causes the text buffer to be output. All text written into the output buffer is then written into the output file and the file closed. Control then returns to either the Keyboard Monitor or Command Decoder, depending on the /A Switch.
.= or .: /= or /:		By typing these characters the user can obtain the current line number (.=) and the last line number in the text buffer (/=). The number is printed by the Editor immediately after the user types the equal sign. (The colon character is equivalent to the equal sign.)

4.5 EDITOR TEXT BUFFER

In text mode, the Editor performs I/O operations on the text within the text buffer.

Approximately 5600 decimal characters can be accommodated in the text buffer. Any changes, additions, character searches, and deletions use available buffer space; i.e., changing line 1 causes a new line to be generated while the old line is not deleted from the physical text buffer (the line is no longer a logical part of the text within the buffer, however). Deleting lines of text, similarly, does not reclaim buffer space used by the deleted lines. The user should, therefore, avoid completely filling the buffer so that useful editing can be performed.

When the point where only 256 decimal locations are available in the buffer is reached, a warning bell is rung. Whenever another carriage return is encountered on input, control returns to command mode and the Teletype bell rings. Under these conditions, the user must again give the command and the Editor accepts one line at a time. When the absolute end of the buffer is reached, no more characters are added to the buffer. Any attempt to add characters results in a ? and return to command mode.

4.6 SEARCH MODE

There are two types of search available in the PS/8 Editor. The first is the standard character search command which is of the form:

```
        #S
        #n S
or      #m,n S
```

The search command searches the entire text buffer or the line(s) indicated for the search character. The search character is typed by the user after the RETURN key which enters the command. The search character 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. Printing stops after the search character is printed.

When the search character is found, the user has the following options:

Type	Result
CTRL/G (BELL rings)	Change the search character to the next character typed.
CTRL/FORM	Continue searching for character.
RETURN key	End line here, deleting any following text on that line.
LINE FEED key	Make two lines out of the current line by inserting a carriage return character at this point.
RUBOUT key	Delete characters from this line. Each RUBOUT echoes a backslash (\) for each character deleted.

4.6.1 Intra-buffer Character String Search

The second type of search available is the character string search. This 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 then be used to further edit the text. There are two types of string search available: Intra-buffer search and inter-buffer search.

The intra-buffer search will stop when all text in the current buffer has been scanned. If the string is not found, a ? is printed and control returns to command mode. If the string is found, the line number is put into the current line counter, and control stays in command mode, waiting for further instructions. Searching for a string merely furnishes a line number which can be used with the other Editor commands. This provides a useful framework for editing, as one no longer need 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 Editor's #. The user then types the string to be found (up to 20 characters long, 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 and causes the search to be performed. Use of the single

quote starts the search at line 1 of the text buffer. Use of the double quote (") instead of the single quote causes the search to begin at the current line + 1. Use of ' and " as command elements prohibits their use in the search string.

For example, if the text buffer contains:

```
ABC DEF GHI
1A2B3C4D5E6
STRINGOABCDEFG
.
.
.
```

and the user wants to list the line that contains RINGO. This could be done by typing:

```
#$RINGO'L
```

and the RETURN key in response to the # printed by Editor ($ is printed by the ALT MODE key). The search begins with line 1 and continues until the string is found. The current line counter is set to the line in which the string occurred, the line is printed as follows:

```
STRINGOABCDEFG
```

and control returns to command mode, awaiting further commands.

If the user wanted to find the next reference to RINGO, 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 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 use can type:

```
#$'
```

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

The properties of ' and " allow for easy and useful editing. Assume the following text:

```
     TAD GO
     DCA GAK
DUM, OSR
     SNA CLA
     CIF 2Ø     /NEW FIELD
     .
     .
     .
```

In order to change the CIF 20 to CIF 10, the user can give the following command:

```
#$DUM,'$CIF 2Ø"C
CIF 1Ø      /NEW FIELD
(type CTRL/FORM)
```

The above set of instructions to the Editor caused a search for the line beginning with DUM, and began the search with line 1. Then a search was made for CIF 20 starting from the line after the line containing DUM. The line number of the line containing CIF 20 is the current line number when the C (Change) command is given. The user then changes the line to the correct instruction.

In using this search feature, the result is a line number. Thus any operations which can be done by explicitly specifying a line number can be done by specifying a string.

For example,

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

will list the fourth line after the first occurrence of 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, use strings first. For example:

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

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

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

4.6.2 Inter-Buffer Character String Search

The inter-buffer search scans the current text buffer for an object 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 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 line number of the first occurrence of the string.

The form of the command is as follows:

```
#J                  Where # is printed by Editor
$GONZO'             $ is printed by Editor auto-
#.=24               matically. Search proceeds,
                    first occurrence of GONZO
                    was line 24 of the current
                    buffer.
```

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                  Search for the string GONZO again
#.=1Ø6              starting at the current line + 1.
```

After the J or F commands have processed the entire input file, it is necessary to execute either an E or Q command to close the output file. If this is not done, the file will be deleted by the monitor. J and F give a "?" error message if no output file exists.

4.7 ERROR MESSAGES

Minor errors are indicated by a question mark at the left margin of the Teletype paper. A ? is caused by an Editor command string error, an attempt to execute a text type command (R, Y, P, N, etc.) without assigning a device, or a search for an unfound string.

Major errors cause the system to leave the Editor and return to the Keyboard Monitor. These messages are of the form:

```
? n ↑C
```

where n is an error code and the ↑C indicates that control has passed to the Keyboard Monitor. These error codes and their meanings are listed in Table 4.3.

TABLE 4.3

Editor Error Codes

Error Code	Meaning
ø	Editor failed in reading a device. Error occurred in device handler, most likely a hardware malfunction.
1	Editor failed in writing onto a device. Generally a hardware malfunction.
2	File close error occurred. For some reason the output file could not be closed; the file does not exist on that device.
3	File open error occurred. This error occurs if the output device is a read only device or if no output file name is specified on a file-oriented output device.
4	Device handler error occurred. The Editor could not load the device handler for the specified device. This error should never occur.

When the output device is full and a write is attempted on that device, an error occurs. The output file is closed, the message

```
FULL
*
```

is printed, and control returns to the Command Decoder for a new set of I/O specifications. The new output file will contain the text that would not fit on the output device, and any further editing the user wishes to do.

1) The contents of the text buffer are retained through this procedure. Thus, no text will be lost if this error occurs.

2) If no output file is specified when control returns to Command Decoder, the Editor returns to Command Decoder again. This continues until an output device is specified.

   CAUTION: Specifying an improper output device (such as PTR:) will cause a fatal error, and the output buffer will be destroyed.

3) If the output device is valid, Editor will continue the operation which filled the old file, putting all output into the new output file. After editing is completed, the two files should be concatenated with PIP or EDITOR, as follows:

```
.R EDIT
*OUT←IN
#Y
#J
$STRING'
FULL
*DTA3:OUT2←
#.L
        TAD STRING
#.D
#E
FULL
*DTA4:OUT3←
.
```

   At this point, the output file is the concatenation of DSK:OUT,DTA3:OUT2, and DTA4:OUT3. When the output file is split like this, the split may occur in the middle of a line. Therefore, never try to edit the output files separately as the split lines are lost. First combine the files with PIP, in this case, as follows:

```
.R PIP
*DTA2:OUT DSK:OUT,DTA3:OUT2,DTA4:OUT3
```

CHAPTER 5

PAL-8 ASSEMBLER

PAL-8 is the PS/8 version of 8K PAL-D on the Disk Monitor System. For a detailed description of PAL-8, read the information on 8K PAL-D in Programming Languages. PAL-8 has, in addition to the features of 8K PAL-D, five extra pseudo-ops: FILENAME, DEVICE, IFNDEF, IFNZRO, and FIXMRI.

A complete listing of PAL-8 permanent symbols is located in Appendix C.

5.1 CALLING AND USING PAL-8

The user can call PAL-8 from the system device by typing:

```
.R PAL8
```

where the dot was printed by the Keyboard Monitor. The system replies by activating the Command Decoder which, in turn, prints a star at the left margin of the teleprinter paper.

As input to the Command Decoder, the user types the name of a binary output file followed by that of a listing output file. The left angle bracket is followed by 1 to 9 input files and various options. For example:

```
*BINARY,LISTING<INPUT
```

A null output file indicates no output of that type is to be generated.

The assembler prints on the teleprinter any error messages encountered in the program. Typing CTRL/O at the keyboard during an assembly will cause error messages not to be printed on the Teletype; messages are still printed in the output file.

If extensions to the file name are omitted, the following assumed extensions are assigned:

File Type	Extension
Input	.PA
Binary output	.BN
Listing output	.LS

The following options can be indicated in the line typed to the Command Decoder.

TABLE 5.1

PAL-8 I/O Options

Option	Meaning
/S	Omit the symbol table normally generated with the listing (applicable only if a listing file is specified).
/N	Generate the symbol table, but not the listing (applicable only if a listing file is specified).
/L	Call the Absolute Loader at the end of the assembly and load the binary file (only applicable if a binary file is specified).
/G	Call the Absolute Loader, load the binary file, and begin execution at location 200.
/D	Generate a DDT-compatible symbol table.
/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).
/H	Generate non-paginated output. Header, page numbers, and page format are suppressed.
/K	Used in assembling very large programs, causes systems containing 12K or more of core to use field(s) 2 and up as symbol table storage.

When the /L or /G option is specified, the user can also include any option to the Binary Loader in the I/O specification line for PAL-8, such as the = starting address option.

5.2 EXAMPLES OF I/O SPECIFICATION STRINGS

Example 1:

```
.R PAL8
*PTP:,LPT:<SOURCE
```

The above lines cause the PAL-8 assembler to be brought into core from the system device and the program SOURCE.PA (or SOURCE) is assembled. The binary output of the assembly is put onto the paper tape punch and the listing and symbol table on the line printer.

Example 2:

```
.R PAL8
*,LISTIN<PROG /S
```

The above I/O specification line causes PAL-8 to assemble PROG.PA (or PROG), putting the listing only into the file LISTIN.LS on DSK. No binary output is generated.

Example 3:

```
.R PAL8
*BIN<INPUT.XY /G=6ØØ
```

The above I/O specification line assembles INPUT.XY, putting the binary output into BIN.BN and then calls the Binary Loader which loads BIN.BN and starts it at 00600. (=600 is an option to the Binary Loader specifying the starting address).

Example 4:

```
.R PAL8
*DTA1:PROG
```

The above lines will assemble file PROG from device DTA1 checking for errors, which are listed on the teleprinter. There are no output files.

5.3 PAL-8 PSEUDO-OPS

In addition to the 8K PAL-D pseudo-ops, PAL-8 also has the following: IFNDEF, IFNZRO, DEVICE, FILENAME, and FIXMRI (as in PAL III).

IFNDEF is similar to IFDEF in 8K PAL-D. The pseudo-op is expressed in the form:

```
IFNDEF symbol <statements>
```

If the symbol indicated has <u>not</u> been previously defined, the statements enclosed in angle brackets will be executed. If the symbol is defined, the statements in the angle brackets are ignored. Any number of statements can be contained in the angle brackets and may consist of several lines of code.

IFNZRO is similar to IFZERO in 8K PAL-D. The pseudo-op is expressed in the form:

IFNZRO expression <statements>

If the evaluated (arithmetic or logical) expression is not equal to zero, assemble the statements within the angle brackets. If the expression is equal to zero, ignore these statements. The expression cannot contain any imbedded spaces and must have a single space preceding and following it. Any number of statements can be contained in the angle brackets and may consist of several lines of code.

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.

DEVICE and FILENAME are used to generate parameters for calling PS/8 I/O routines (see the PS/8 Programmer's Reference Manual, DEC-P8-MEXA-D). They are of the form:

```
DEVICE name
FILENAME name
```

With DEVICE, name can be 1 to 4 alphanumeric characters. The assembler allocates 2 words for name and converts the characters to 6-bit ASCII, filled with zeros on the right.

With FILENAME (or FILENA, which is acceptable), name (or name.extension) can be as many as 8 characters altogether. Four words are allocated for the storage of name in 6-bit ASCII, the first three words are the file name filled with zeros on the right, and the last word is the file extension. For example:

```
L, FILENAME ABC.DA
```

is equivalent to the following coding:

```
L, Ø1Ø2
   Ø3ØØ
   ØØØØ
   Ø4Ø1
```

FIXMRI makes a symbol a memory reference instruction. All FIXMRI

pseudo-ops are of the form:

```
FIXMRI  name=value
```

FIXMRI can appear anywhere in a PAL-8 program. 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 pseudo-op must be repeated for each memory reference instruction to be defined. For example:

```
FIXMRI FADD=1ØØØ
FIXMRI FSUB=2ØØØ
FSQRT=2
FIXTAB
PAUSE
```

When the preceding program segment is read into the Assembler during pass 1, the three symbols listed are added to the permanent symbol table. Notice that FSQRT is not a memory reference instruction. This process is often performed to alter the Assembler's symbol table so that it contains those symbols used at a given installation or by a given program.

The following is a summary of the PAL-8 pseudo-ops. These pseudo-ops are the same as for 8K PAL-D and are described in greater detail in Programming Languages.

TABLE 5.2

PAL-8 Pseudo-ops

Mnemonic Code	Operation
$	Indicates the end of a program, terminates each pass of the assembler.
DECIMAL	Decimal conversion, numeric conversion interprets all numbers input as being decimal numbers.
DEVICE	Used to generate I/O routine parameters. See section 5.3.
EJECT	Causes the listing to jump to the top of the next page. (A page eject is done automatically every 55 lines.)

TABLE 5.2 (Cont'd.)

PAL-8 Pseudo-ops

Mnemonic Code	Operation
ENPUNCH	Causes the assembler to resume or continue binary output.
EXPUNGE	Deletes the entire permanent symbol table, except for pseudo-ops.
FIELD	Causes a field setting to be output on the binary output file which tells the Loader to begin loading information into the new field. For example: FIELD 1 will cause subsequent statements to be loaded into field 1.
FILENAME	Used to generate I/O routine parameters. See section 5.3.
FIXMRI	Defines a memory reference instruction. See section 5.3.
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.
I	Symbolic representation for indirect addressing. For example DCA I ADD
IFDEF	Of the form: IFDEF symbol <statements> If the symbol indicated is previously defined, the statements in the angle brackets are assembled. If undefined, ignore these statements.
IFNDEF	See section 5.3.
IFZERO	Of the form: IFZERO expression <statements> If the evaluated (arithmetic or logical) expression is equal to zero, assemble the statements within the angle brackets. If the expression is non-zero, ignore the statements.
IFNZRO	See section 5.3.
NOPUNCH	Upon encountering this statement, the assembler continues to assemble the code, but ceases binary output. See ENPUNCH.
OCTAL	Octal conversion; numeric conversion is originally set to octal and can be changed back to octal after a DECIMAL pseudo-op has been used.
PAGE	Terminate current page, begin assembly of succeeding instructions on next core page.
PAUSE	Acts as a NOP in PAL-8. PAUSE appears only for compatibility with PAL III and PAL-D.

TABLE 5.2 (Cont'd.)

PAL-8 Pseudo-ops

Mnemonic Code	Operation
TEXT	Text string. Characters are stored in six-bit ASCII, with a printing character used to delimit the string. For example: TAG, TEXT /123*/ the string would be stored as: 6162 6352 ØØØØ
XLIST	Those portions of the source program enclosed by XLIST pseudo-ops will not appear in the listing file.
Z	Optional method of denoting a page zero reference. DCA ADD DCA Z ADD The two statements generate the same code, where ADD is on page 0.
ZBLOCK	Causes the assembler to reserve n words of memory containing zeros, starting at the word indicated by the current location counter. For example: ZBLOCK 4Ø causes the assembler to reserve 40 (octal) words.

PAL8 ARITHMETIC AND LOGICAL OPERATORS

+	plus
-	minus
↑	multiplication
&	logical AND
!	logical OR, inclusive
␣	logical OR, inclusive

5.4 PAL-8 ERROR MESSAGES

The error messages for PAL-8 are the same as those for 4K PAL-D and 8K PAL-D. The format of the error messages is

ERROR CODE ADDRESS

where ERROR CODE is a two 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. The programmer should examine each error indication to determine whether correction is required.

TABLE 5.3

PAL-8 Error Codes

Error Code	Explanation
BE	Two PAL-8 internal tables have overlapped--This situation can usually be corrected by decreasing the level of literal nesting or the number of current page literals used prior to this point on the page.
DE	Device Error--An error was detected when trying to read or write a device. Control is returned to the Monitor.
DF	Device full--The capacity of an output device has been exceeded; assembly is terminated and control is returned to the Monitor.
IC	Illegal Character--An illegal character was encountered other than in a comment or TEXT field; 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 was 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; a link could not be generated because the indirect bit was already set. For example: *2ØØ TAD I A . . . PAGE A, 724Ø
LD	This message is given if the /L or /G options have been specified and the Absolute Loader cannot be found on the system device.
PE	Current non-zero page exceeded--An attempt was made to: 1. Override a literal with an instruction, or 2. Override an instruction with a literal; this can be corrected by (a) Decreasing the number of literals on the page, or (b) Decreasing the number of instructions on the page.
PH	Phase error--means either: no $ appears at the end of the program, or the < and > used in conditional pseudo-ops do not match.
SE	Symbol table exceeded--Assembly is terminated and control is returned to the Monitor; the symbol table may contain up to approximately 880 user symbols in 8K of core.
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

UTILITY PROGRAMS

The following chapter treats four important PS/8 system programs: PIP, CONVERT, ABSLDR, and ODT. These programs are used to manipulate files and file directories and for loading and debugging programs on the PS/8 system.

6.1 PERIPHERAL INTERCHANGE PROGRAM (PIP)

PIP is used to transfer files between devices; provide directory listings; delete and zero directories; and compress the device directories, eliminating spaces left by the delete option.

6.1.1 Calling and Using PIP

To call PIP from the system device the user types:

```
.R PIP
```

in response to the dot printed by the Keyboard Monitor. The Command Decoder then prints a star at the left margin of the teleprinter paper and waits to receive a line of I/O files and options. PIP accepts up to nine input files and performs output to a single output file.

Since PIP performs file transfers for all file types (ASCII, Image or SAVE format, or binary), there are no assumed extensions assigned by PIP to file names for either input or output files. All extensions, where present, must be explicitly specified.

Following completion of a PIP operation, the Command Decoder again prints a * at the left margin and waits for another PIP I/O specification line. The user can return to the Keyboard Monitor by typing CTRL/C.

The various options allowed on a PIP I/O specification line are detailed in Table 6.1. Either /A, /B, or /I is generally indicated for each transfer; if none of these are used the system proceeds as though /A had been typed.

TABLE 6.1

PIP I/O Options

Option	Meaning
/A	Transfer files in ASCII mode. The file is modified as it is copied: embedded blank tape and RUBOUTs are deleted and leader/trailer code is reduced to a standard length. PIP may also do some editing of the input file under control of the /C and /T options (see below).
/B	Transfer files in binary mode (used for absolute and relocatable binary files). Leader/trailer code is reduced to a standard length, but the checksum is not recalculated. NOTE: If several absolute binary files are combined into one, the /S option must be given to the Absolute Loader in order for them to load properly. The Linking Loader will not load combined files at all.
/I	Transfer files in image mode. Used to transfer core image (SAVE format) files, and any other files which do not fall into either ASCII or binary categories. Nothing is done to alter the input file.
/Z	Zero directory of output device before file transfer. Before using a DECtape for the first time, the /Z option should be used with no input files to create an empty file directory. For example: .R PIP *DTA2:/Z<
/D	Delete old copy of the output file before the transfer. If /D is not used, the old copy would not be deleted until all data are transferred. For example: DTA1:FILE/D←NEW FILE /D may also be used to delete up to 3 files at a time by specifying the files to be deleted as output files and not specifying any input files. For example: *OLDABC,DTA3:FILE5/D< deletes OLDABC from DSK and FILE5 from DTA3.
/C	Eliminate trailing blanks. Valid in ASCII mode only.

TABLE 6.1 (Cont'd.)

PIP I/O Options

Option	Meaning
/T	Perform conversions of special characters as follows:

Character	Is Converted To
TAB	enough spaces to reach the next TAB stop (every eighth position)
Vertical TAB	5 line feeds
FORM FEED	9 line feeds

/T option is valid in ASCII mode only.

=n	Save n extra words per file entry in the directory to contain descriptive information about the file. For use with the /Z and /S options only. Typing =1 allows the date of the file creation to be automatically stored in the directory.
/L	Lists the directories of the input devices onto the output file starting at the file specified. Notice that in this case the input file itself is not transferred, only the directory.
/F	Lists directories in short form (file names only).
/E	Lists directories in extended form (the lengths of empty files are also listed).
/S	Moves the contents of the input device onto the output device, eliminating all free files. This combines all free files into a single contiguous block instead of many fragments. Programs are packed together when the files are transferred from one device onto another or onto the same device.
/G	If errors occur during file transfer, ignore them and keep copying.

No data transfer occurs if there are no input files specified. Thus, /Z can be used to zero a directory, and /D can be used to delete a permanent file entry without creating a file.

For the three directory listing options (/L, /E, /F): if no output device is specified, the teleprinter is assumed. If no input device is specified, DSK: is assumed.

Whenever the /S option is used, PIP prints the message:

ARE YOU SURE?

to which the user responds with the letter

```
Y
```

if he does wish the compression (and, possibly transfer) to occur. Typing any other character aborts the /S option.

6.1.2 Examples of PIP I/O Specification Commands

The following are legal command strings to PIP. When PIP has completed an operation, control returns to the Command Decoder for additional input to PIP.

Example 1:

```
.R PIP
*SYS:BLACK<PTR:
```

The above command transfers a tape from the high-speed reader to a file on the system device under the name BLACK. PIP assumes that the input tape is in ASCII format. (The system returns to the Command Decoder; the R PIP command need only be given once.)

Example 2:

```
*DTA3:MERGE<DTA1:FILE1,FILE2
```

merges the ASCII files FILE1 and FILE2 on DTA1 into one ASCII file, MERGE on DTA3.

Example 3:

```
*BIN.BN<PTR:/B
```

reads a binary paper tape from the high-speed reader and creates a binary file BIN.BN on the device DSK.

Example 4:

```
*SYS:GAG.SV<PAL8.SV/I
```

transfers the core image file PAL8.SV from the device DSK to GAG.SV, on the system device.

Example 5:

```
*TTY:</E
```

produces an extended listing of the device DSK on the Teletype. An extended listing contains all files with their associated lengths, and all "holes" in the directory. For example, an extended listing might appear as follows (the current date is printed before the file listing):

```
9/14/7Ø
ABSLDR.SV      4        8/21/7Ø
PIP.SV        1Ø        8/21/7Ø
APPLE         17        9/12/7Ø
<EMPTY>        3
JACK          14        9/Ø8/7Ø
<EMPTY>      52Ø

523 FREE BLOCKS
```

The file lengths are decimal numbers (not octal), as is the number of free blocks. The date of file creation is printed if at least one additional information word is present in the directory (see section 6.1.3).

Example 6:

```
*/F
```

produces a directory listing of file names only. Thus, the above directory would appear on the teleprinter as follows:

```
9/14/7Ø
ABSLDR.SV
PIP.SV
APPLE
JACK

523 FREE BLOCKS
```

Example 7:

```
*LPT:<DTA2:FETCH/L
```

produces a listing of the DTA2 directory on the line printer; however, the files that occur before FETCH are not listed. The

/L option gives the regular listing which includes the file name and extension length, and date (if a date is contained in the directory). Empty files are not indicated in the listing.

6.1.3 Additional Information Words in File Directories

If a device has any additional information words specified in its directory, PS/8 automatically enters the last date specified in a DATE command into the first of the additional information words when a file is created on that device. (See Section 2.3.9 describing the DATE command.)

Dates put into these additional words appear in directory listings. Words after the first are not currently used by the PS/8 system.

The additional words must be specified by a /Z=n or /S=n option. The number of additional words can be changed once assigned to a device by compressing the device onto itself.

When using /S to transfer from a device which contained no additional information words, if those words are specified in the output device directory the date entries are garbled. This can be fixed by copying each file onto itself, using the PIP /I and /D options as follows:

```
device:file<device:file(ID)
```

NOTE

The system is initially created with one additional information word in the file directory.

6.1.4 PIP Error Messages

TABLE 6.2

PIP Error Messages

Message	Meaning
NO ROOM FOR OUTPUT FILE	Self explanatory; either room on device or room in directory is lacking.
LINE TOO LONG IN FILE # n	In ASCII mode a line has been found greater than 140 characters. Be sure the transfer is really in ASCII mode. n is the number of this file in the input file list.
OUTPUT ERROR	Output error, possibly a write locked device, parity error, or attempt to output to a read-only device.
ERROR DELETING FILE	The user tried to delete a file that does not exist.
INPUT ERROR, FILE # n	An input error occurred while reading file number n in the input file list.
CAN'T OPEN OUTPUT FILE	Output file on read-only device, or else there is no name specified for the file.
DEVICE # n NOT A DIRECTORY DEVICE	Error message given by directory listing options when an incorrect device designation is made. n is the number of the device in the input list.
PREMATURE END OF FILE, FILE # n	/B option, incomplete binary input of file n in the input file list (ran out of input before finding trailer tape, for example).
ILLEGAL BINARY INPUT, FILE # n	Self explanatory, n is the number of the file in the input file list.
BAD DIRECTORY ON DEVICE # n	Directory listing error, the system is trying to read the directory of a blank device; where n is the number of the device in the input list.

TABLE 6.2 (Cont'd.)

PIP Error Messages

Message	Meaning
DIRECTORY ERROR	/S error, an error has occurred while reading or writing the directory during an /S option. The option is aborted; output is likely to be garbled.
IO ERROR - CONTINUING	/S error, error in copying a file. The /S option continues.
ARE YOU SURE?	/S message, respond with a Y if you are sure you want to compress the files.
SORRY - NO INTERRUPTIONS	/S message given if ↑C is typed while compressing a device onto itself. The /S option continues.
NO ROOM - CONTINUING	/S message given when the output device cannot contain all of the files on the input device. The message is printed once for each file which will not fit onto the output device.

6.2 ABSOLUTE BINARY LOADER (ABSLDR)

The Absolute Binary Loader is used to load the binary output from the PAL-8 Assembler. The input files are loaded according to the options specified, and a core control block is constructed. (For a description of the core control block, see Section 2.3.3).

6.2.1 Calling and Using ABSLDR

The user calls the Absolute Binary Loader from the system device by typing

```
.R ABSLDR
```

in response to the dot printed by the Keyboard Monitor. The system responds by printing a * at the left margin. The user then types an input line to ABSLDR, indicating input files and any options desired. ABSLDR does not recognize any output files, since the purpose of the loader is to load and start binary output files.

The standard input devices for ABSLDR are: PTR, DTAn, DSK, and SYS. Any other device which can contain absolute binary files can be used as an input device if a device handler exists. TTY should not be used, as the binary code may appear to the TTY handler as control characters.

ABSLDR normally accepts absolute binary files. Relocatable files must be loaded with the Linking Loader (see Section 7.3). Save (.SV) format files can be loaded with the I option. If no extension to the input file name is typed, ABSLDR assumes the .BN extension. Up to nine input files are allowed. If more than one program is present in a file, only the first program is loaded. (This feature allows ABSLDR to ignore any noise characters which might be caused by reading over the end of a paper tape.)

By typing the RETURN key at the end of an input specification line, the loader is signalled that more input is to be given on the next line. If the ALT MODE key is used as a line terminator, no more input is expected, the Command Decoder is not recalled, and control returns to the Keyboard Monitor. For example:

```
.R ABSLDR
*DTA1:FILE1,FILE2,FILE3,FILE4
*PTR:$
.
```

The preceding lines cause FILE1, FILE2, FILE3, and FILE4 to be loaded at their absolute locations in core from DECtape 1. Then a file is to be read from the high-speed paper tape reader. The $ character is printed by the ALT MODE key which indicates a return to the Keyboard Monitor.

The various options accepted by ABSLDR are detailed in Table 6.3.

TABLE 6.3

ABSLDR I/O Options

Option	Meaning
/8	Used when locations Ø-1777 of field Ø are not being used by the program. Eliminates extra DECtape motions to save those locations, hence saves time. For additional information, see the section 2.3.3 or the PS/8 Programmer's Reference Manual (DEC-08-MEXA-D).
/9	Similar to /8, used when locations Ø-1777 of field 1 are not to be saved.
/I	Treat the input file(s) as a core image file to be overlaid with the input of succeeding lines. If this option is not used in the first command line, it cannot be used unless ABSLDR is recalled from the Keyboard Monitor level. The /I option can be used to make patches to an already saved program without reassembling the entire program.
/R	Resets internal core map of ABSLDR to look as though nothing has been loaded into core.
/S	Load all binary programs in the specified input file(s) (instead of loading only the first program in each file, which is normally done).
/G	Start program execution upon finishing the loading procedure. Normally, control returns either to Monitor or Command Decoder (depending on the terminator key). If /G is specified, control is given to the program just loaded. The starting address is assumed to be 200 unless specified in the input string. Control stays with the user's program until he releases it to the Monitor from within his program. No automatic return to Monitor or the Command Decoder occurs.
/n	Where n is an integer, forces loading of all files specified on this input line into field n.
=n	Set the starting address of the program in core to n, where n is a 5 digit octal integer.

6.2.2 Examples of Input Lines

Example 1:

```
.R ABSLDR
*SYS:RPOG.SV /I
*DTA1:PATCH$
.SAVE SYS:PROG
```

The above commands load the core image file PROG.SV and then overlay part of that program file with a binary patch from DTA1. Control then returns to Monitor at which time the user saved the patched program on the system device.

When using the /I option, the starting address and Job Status Word of the core image being loaded are ignored by the Loader. The user must specify starting address and contents of Job Status Word.

As another example, the user could overlay PIP with a binary patch which will not change its starting parameters:

```
.R ABSLDR(CR)
*PIP.SV/I(CR)
*PTR:=13ØØØ(89)$
.SAVE SYS PIP
```

This could also be done using an explicit SAVE, i.e.,

```
.R ABSLDR
*PIP.SV/I(CR)
*PTR:$
.SAVE SYS PIP;13ØØØ=6ØØ3
```

Example 2:

```
.R ABSLDR
*PTR:(89)/G$
```

One binary tape is loaded from the high-speed paper tape reader. Areas ØØØØØ-Ø1777 and 1ØØØØ-11777 of core are not used by the program. The starting address of the program is considered to be 200 and control transfers to the user program.

6.2.3 ABSLDR Error Messages

In each case control returns to the Command Decoder and the user can try the procedure again, or reset the loader (using the /R option) and try again using different inputs.

TABLE 6.4

ABSLDR Error Messages

Message	Meaning
I/O ERROR FILE # n	An I/O error has occurred in input file number n
BAD INPUT, FILE n	Attempt to load non-binary file as file number n of the input file list, or a non-core image with /I option
BAD CHECKSUM, FILE # n	File number n of the input file list had a checksum error
NO INPUT	No input file was found on the designated device
NO /I!	Use of /I is prohibited at this point

6.3 ODT

ODT, Octal Debugging Technique, allows the programmer to run his binary program on the computer, control its execution, and make alterations to his program by typing instructions at the keyboard.

Usage of ODT is described in Chapter 6, Introduction to Programming 1970. The presentation of ODT in this section is a summary of the commands and the application of ODT to the PS/8 system.

6.3.1 Calling and Using ODT

As explained in section 2.3.6, ODT is called into use by typing:

```
.ODT
```

where the dot was printed by the Keyboard Monitor. Before calling ODT, the user should have a running version of his program in core. When ODT is used, none of the user's core is absorbed by the running of ODT. ODT carefully preserves the sections of the

user's program which it occupies in core on the system device and swaps it back into core as necessary. ODT is invisible to the user and does not limit his program. Whenever using the breakpoint feature of ODT, locations 4, 5, and 6 of the memory field in which the breakpoint is set, are used by ODT.

ODT should not be used to debug programs which use interrupts.

Typing CTRL/C causes ODT to disappear and returns control to the Keyboard Monitor from where the user can save the program on any device.

6.3.2 Summary of ODT Commands

Addresses can be five digits long on input and are printed as five digits.

The following is a brief summary of the ODT commands. Additional notes follow the summary.

TABLE 6.5

ODT Command Summary

Command	Meaning
nnnnn/	Open location designated by the octal number nnnnn, where the first digit represents the memory field. ODT prints the contents of the location, a space, and waits for the user to enter a new value for that location or close the location.
/	Reopen latest opened location.
nnnn;	Deposit nnnn in the currently opened location, close that location and open the next sequential location for modification. A series of octal values can be deposited in sequential locations through use of the ; character. Multiple ;'s skip a memory location for each ; typed and prepare to insert subsequent values beyond the one(s) skipped.
RETURN key	Close the previously opened location.
LINE FEED key	Close location and open the next sequential one for modification and print the contents of that location.

TABLE 6.5 (Cont'd.)

ODT Command Summary

Command	Meaning
n+	Open the current location plus n for modification and print the contents of that location.
n-	Open the current location minus n for modification and print its contents.
↑ or ^ (up-arrow or circumflex)	Close location, take contents of that location as a memory reference and open the location referenced, printing its contents.
← or _ (back-arrow or underline)	Close location, take contents of that location as a 12-bit address and open that address for modification, printing its contents.
nnnnnG	Transfer control of program to location nnnnn, where the first digit represents the memory field.
nnnnnB	Establish a breakpoint at location nnnnn, where the first digit represents the memory field. Only one breakpoint is allowed at any given time.
B	Remove the breakpoint.
A	Open for modification the location in which the contents of the accumulator were stored when the breakpoint was encountered.
L	Open for modification the location in which the contents of the link were stored when the breakpoint was encountered.
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 LINE FEED LINE FEED	Open the search mask, initially set to 7777, which can be changed by typing new value. Open the lower search limit (four octal digits) Open the upper search limit (four octal digits)
nnnnW	Search the portion of core as defined by the upper and lower limits for the octal value nnnn. Search can only be done on a single memory field at a time. See the F command.
D	Open for modification the word containing the data field which was in effect at the last breakpoint. Contents of D always appear as multiples of 10_8 - i.e., 10 means field 1, 20 field 2, etc.
CTRL/O	Stop any printing currently in progress.

TABLE 6.5 (Cont'd)

ODT Command Summary

Command	Meaning
F	Open for modification the word containing the field used by ODT in the W (search) command, in the ← and ↑ (indirect addressing) commands, or in the last breakpoint (depending upon which was used most recently. The contents of F are always expressed as multiples of 10_8 (as in the D command).
RUBOUT key	Cancels previous number typed, up to the last non-numeric character typed.

When using the breakpoint feature, the user should keep certain operating characteristics of ODT in mind:

a. ODT keeps track of the TTY flag and restores the TTY flag when it continues from a breakpoint

b. Breakpoint feature uses locations 4, 5, and 6 in the memory field in which the breakpoint is set.

c. The Breakpoint feature of ODT uses the table of user-defined device names as scratch storage, destroying any device names the user may have created. After a session with ODT in which breakpoints are used, the user should give a DEASSIGN command to clear out the user-device name table.

d. Breakpoints must not be set in the Monitor, in the device handlers, or between a CIF and the following JMP instruction.

The user is advised not to use user-defined device names in programs being developed with ODT breakpoints.

ODT uses the Job Status Word of the particular program to determine whether or not swapping occurs. If the program does not use locations Ø-1777 in field 0, less swapping occurs during use of the breakpoint feature.

If the user is typing any amount of a program directly into core (in octal), the core control block of the program may not reflect the true extent of the program. If octal additions are

made below location 2000 in field 0, ODT may give erroneous results. The user can correct this condition by correcting the Job Status Word.

The Job Status Word is location 7746 of field 0. The Job Status Word can be examined and changed with ODT. Location 7745 of field 0 is the 12-bit starting address of the program in core and location 7744 contains the field designation in the form 62n3 where n is the field designation of the starting address.

6.4 PS/8 File Conversion Program (CONVRT)

CONVRT transforms DECtape ASCII files from either 4K Disk Monitor or TSS/8 format to PS/8 format.

6.4.1 Calling and Using CONVRT

To call CONVRT fromthe system device, the user types:

```
.R CONVRT
```

in response to the dot printed by the Keyboard Monitor. CONVRT only recognizes DECtape files as legal input; any other device causes an error message. Only one input file is accepted at a time. A single DECtape file is indicated as input and is put into PS/8 format on the designated output device.

CONVRT can use any device accepting ASCII code for output, as long as it has a device handler within the system. These devices usually include: SYS, DTAn, DSK, LPT, and PTP. Only one output file is specified to CONVRT at a time.

Upon completion of an operation, CONVRT returns to the user for more input by printing another start at the left margin. The user can then input additional commands to CONVRT, or return to the Keyboard Monitor by typing CTRL/C.

The I/O specification line can include one of the following options:

TABLE 6.5

CONVRT I/O Options

Option	Meaning
/T	Input tape is a TSS/8 DECtape. Use the /T option whenever a TSS/8 tape is being converted.
/L	Produce a listing of all ASCII files on the input tape. For a TSS/8 tape, both /T and /L must be issued to obtain a correct listing.
/K	When converting 4K Disk Monitor DECtape files, the file directory is brought into core with the first file conversion specified. Subsequent files from the same input device need not reread the directory. Time can be saved by using the directory already in core to convert later files. The /K option is only valid after the initial conversion of a file, as the initial pass reads the directory into core.

6.4.2 Examples of CONVRT I/O Specification Commands

Example 1:

```
.R CONVRT
*TTY:<DTA2: /L
```

The above commands cause CONVRT to be started and all ASCII files present on DTA2 are listed. CONVRT assumes 4K Disk Monitor input format, since /T is not specified. The listing of the file names is put on the teleprinter.

Example 2:

```
.R CONVRT
*LPT:<DTA4: (LT)
```

The above lists all ASCII files present on DTA4 onto the line printer. TSS/8 format is assumed.

Example 3:

```
.R CONVRT
*DTA1:PROG←DTA2:DEMO
*LPT:<DTA2:JOB /K
```

The above commands call CONVRT from the system device, and convert the 4K Disk Monitor file DEMO on DTA2 to a PS/8 file PROG, on DTA1. Now, to convert another file from the same device,

the user typed the last line, which produces a listing of the 4K Disk Monitor file JOB on DTA2 and puts the listing of the file on the line printer. For the second operation, CONVRT uses the Disk Monitor directory already in core (/K option).

6.4.3 CONVRT Error Messages

Any error made while using CONVRT causes a message to be printed and the Command Decoder recalled for a new, corrected, I/O string.

TABLE 6.6

CONVRT Error Message

Message	Meaning
FILE OPEN ERR	An output file could not be opened on the the specified device
OUT DEV FULL	There is no more room on the output device. Something must be deleted on the output device before CONVRT can work properly.
INPUT DEV WRONG	The input device specified is not a DECtape, or SYS was specified as input.
INPUT READ ERR	The DECtape read routine detected some error while reading the input tape.
IN FILE NOT FOUND	The input file was not found, or none was specified.
BAD EOF	A zero link was detected before the logical End of File. The output file is closed at this point.
FILE CLOSE FAILED	An error occurred in closing the output file.
OUTPUT WRITE ERR	Device handler detected an error in transferring data.
OUT DEV HANDLER ERR	Output is inhibited, usually because no output device has been specified.

CHAPTER 7

THE 8K FORTRAN SYSTEM

7.1 THE 8K FORTRAN COMPILER

The 8K FORTRAN Compiler is discussed in detail in Chapter 15 of Programming Languages. For those users not familiar with DEC's 8K FORTRAN, it is suggested that they first read that chapter. This section describes the differences and summarizes the 8K FORTRAN language as implemented on the PS/8 system. PS/8 FORTRAN is an improved version of the old 8K FORTRAN, including such features as Hollerith constants, implied DO loops, chaining, mixing of SABR and FORTRAN statements, and device independent I/O.

7.1.1 Calling and Using the 8K FORTRAN Compiler

The user calls the FORTRAN Compiler by typing:

```
.R FORT
```

in reply to the dot generated by the Keyboard Monitor. When the Command Decoder prints a star at the left margin, the user types the appropriate I/O files and any of the acceptable specification options allowed for 8K FORTRAN.

The line to the Command Decoder consists of 0 to 3 output files; the first file holds the binary output with the assumed extension .RL, the second file the listing with the assumed extension .LS, and the third file the Linking Loader output (with /M, /U, or /P Linking Loader options, explained in Section 7.3.1) having the assumed extension .MP. One to 9 input files are possible with FORTRAN (although ordinarily only 1 is used). For example:

```
*BINARY,LISTING,OUTPUT<INPUT (options)
```

The default extension for FORTRAN input files is .FT and the Compiler produces an output file named FORTRN.TM on the system device for input to the 8K SABR Assembler. The compiler automatically calls SABR after compiling. The FORTRN.TM file is deleted by SABR, unless the /K option is specified. (The /K option indicates that the system is to keep the file FORTRN.TM

as a permanent file.) It is also possible to have the system automatically load or automatically load and execute the output of the SABR assembly; this is done by specifying the /L or /G options, respectively.

SABR outputs a file in relocatable binary format into the specified binary output file. If a binary output file is not specified and the /L or /G option is given, then the binary output goes into a file called FORTRL.TM on the system device. If /L and /G are both absent, a null binary output file indicates that no binary output is to be generated. A SABR listing is not generated if a listing output file is not specified.

FORTRAN only assembles one main program or subroutine per call. A job with multiple programs must be run by compiling each routine separately and combining them with the Linking Loader.

TABLE 7.1

FORTRAN I/O Options

Option	Meaning
/L	Load, but do not start execution. Call the Linking Loader at the end of the assembly and load the specified binary file. If a binary output file is not specified, then the temporary file FORTRL.TM is loaded into core and deleted from the file device. The loader then either returns to the Keyboard Monitor with a core image in core, or asks for more input, depending on whether an ALT MODE or RETURN key terminated the input line.
/G	Load and execute the file. Call the Linking Loader, load the binary output file and execute that file in core. If a binary output file is not specified, then FORTRL.TM is loaded into core and deleted from the file device. If a starting address is not specified (using the options described under the Linking Loader in Section 7.3.1) control is sent to the program entry point MAIN (FORTRAN Compiler gives this name automatically to the main program).
/K	Keep the file FORTRN.TM as a permanent file instead of using it as input to SABR and then deleting it. Used whenever it is desirable to edit the output of the FORTRAN Compiler for later assembly.

The /N and /S options to the SABR Assembler can also be specified to the FORTRAN Compiler. See Section 7.2.1 for details.

Options to the Linking Loader other than /L can also be used (see Section 7.3.1).

7.1.2 Examples of FORTRAN I/O Specification Commands

Example 1:

```
.R FORT
*DTA1:TEST /G
```

The input file TEST.FT (or TEST) on DTA1 is compiled, the output stored in FORTRN.TM on the system device, and SABR is called. SABR uses FORTRN.TM as input and outputs the assembled file into FORTRL.TM, deleting the old FORTRN.TM. The /G option specifies that the Linking Loader then loads FORTRL.TM, deletes FORTRL.TM upon loading, and sends control to the entry point MAIN.

Example 2:

```
.R FORT
*,,LPT:<INPUT/L/M
```

The FORTRAN Compiler compiles and SABR assembles the file DSK:INPUT.FT (or INPUT), outputting the binary file in SYS:FORTRL.TM. The Linking Loader is automatically called (/L) to load SYS:FORTRL.TM into core and delete that file from SYS. The Linking Loader puts a full loading map on the LPT device (/M). The Loader then asks for another command string. If the line had terminated with an ALT MODE key instead of the RETURN key, control would have returned to the Keyboard Monitor after loading.

Example 3:

```
.R FORT
*BINARY,LPT:<MATRIX.AB /N/K
```

The input file MATRIX.AB on DSK is compiled and output into SYS:FORTRN.TM. SABR is called and assembles SYS.FORTRN.TM, putting the relocatable binary output into DSK:BINARY.RL and the symbol table only (/N) on the LPT device. The /K option causes SYS:FORTRN.TM to be kept as a permanent file.

Example 4:

```
.R FORT
*DTA5:SOURCE /L
```

The file SOURCE on DTA5 is compiled, assembled, and loaded, but not executed.

Example 5:

```
.R FORT
*DTA1:PROG,PTP:,PTP:<DTA1: PROG(NMG)
```

For those users with DECtape systems, keeping the source program on a non-system DECtape and putting the binary on a non-system DECtape gives the best possible results in terms of minimizing tape motion.

7.1.3 FORTRAN Language Elements

7.1.3.1 FORTRAN Constants

8K FORTRAN has three types of constants, integer constants, real constants, and Hollerith constants. Integer constants are represented by a digit string of one to four decimal digits with an optional sign and without a decimal point. Integer constants must fall within the range -2047 to +2047. For example:

```
47
+47          (+ sign is optional)
-2
0434         (leading zeros are ignored)
-0           (same as 0 or +0)
```

Real constants are represented either as a decimal number or in exponential form. A real number can have any number of digits, but only the leftmost eight digits are significant (appear in the compiled version). Real constants must fall in the range $\pm$ 1.7 x 10^{38}. For example:

```
+4.50        (+ sign is optional)
4.50
-2361.008
-3.0E14      (same as -3.0 x 10^14)
```

A Hollerith constant is a string of up to 6 characters (including blanks) enclosed in single quotes. A Hollerith constant is treated like a real constant, except that Hollerith constants cannot be used in arithmetic expressions other than for simple equivalence (A=B). Any character except the quote character itself can be used in a Hollerith constant. For example:

```
'MOM'
'A+B=C'
'5 & 10'
```

7.1.3.2 FORTRAN Variables

FORTRAN variables can be integer or real, scalar or array, as defined in standard FORTRAN practices (see Programming Languages for greater detail, if necessary). For example:

LOW	integer variable, scalar
MAX(I,J)	integer variable, array
I	integer variable, scalar
G2(2)	real variable, array
ALPHA(M,N)	real variable, array
SIGMA	real variable, scalar

The first five characters are interpreted as defining the variable name, the rest are ignored.

CAUTION

> Programs containing subscripting compiled with the new FORTRAN compiler (on or after October 1970) will not run with a FORTRAN library previous to October, 1970. A new subscripting algorithm has been used with the new FORTRAN compiler which occupies 50% less core space and allows the user to run an up to 35% larger program (depending on the amount of subscripting used) than with the earlier 8K FORTRAN. Similarly, programs containing subscripting developed with the new FORTRAN compiler will not run under the old FORTRAN system.

Without parentheses, algebraic operations are performed in the following descending order:

**	exponentiation
-	unary negation
* and /	multiplication and division
+ and -	addition and subtraction
=	equals or replacement sign

Parentheses are used to change the order of precedence; an operation enclosed in parentheses is performed before its result is used in other operations. In the cases of equal precedence, the calculations are performed from left to right.

Integers and real numbers can be raised to either integer or real powers.

7.1.3.3 FORTRAN Functions

The FORTRAN Library of functions is loaded on the system device when the PS/8 system is built. The Library must be on the system device in order to use any of the function calls. Once the function(s) are present in the system, the user need only use the standard function call in his program in order for that function to be computed. Table 7.2 contains the list of the FORTRAN function Library.

TABLE 7.2

8K FORTRAN Function Library

Function	Definition	Type of Argument(s)
ABS(x)	the absolute value of x	real
IABS(x)	the absolute value of x	integer
FLOAT(x)	convert x from integer to real format	integer
IFIX(x)	convert x from real to integer format	real
IREM(Ø)	remainder of last integer divide is returned	integer
IREM(x/y)	remainder of x/y is returned	integer
EXP(x)	exponential of x, e^x	real
ALOG(x)	natural logarithm of x, $\log_e x$	real
SIN(x)	sine of x, where x is given in radians	real
COS(x)	cosine of x, where x is given in radians	real
TAN(x)	tangent of x, where x is given in radians	real
ATAN(x)	arctangent of x, where x is given in radians	real
SQRT(x)	square root of x is returned	real
IRDSW(Ø)	read the console switch register, returning a decimal equivalence of the octal integer in the switch register. The switch register can be set before executing the FORTRAN program; or, using the PAUSE statement, during execution.	integer

In general, floating-point arithmetic calculations are accurate to seven digits with the eighth digit being questionable. Subsequent digits are not siginficant even though several may be printed to satisfy a field width requirement. Results of function operations are accurate to six decimal places.

The floating-point arithmetic routines check for both overflow and underflow. Overflow causes the FPNT error message to be printed and program execution terminated. Underflow is detected but does not cause an error message; the arithmetic operation involved yields a zero result.

Integer arithmetic operations do not check for overflow. For example, the sum 2047+2047 yields a result of -2. For more information, refer to Chapter 1 of Introduction to Programming 1970 or any text on binary arithmetic.

Zero raised to a power of zero yields a result of 1. Zero raised to any other power yields a zero result. Numbers are raised to integer powers by repetitive multiplication. Numbers are raised to floating-point powers by calling the EXP and ALOG functions. A negative number raised to a floating-point power does not cause an error message but uses the absolute value of the negative number. Thus, the expression (-3.0)**3.0 yields a result of +27.

7.1.4 FORTRAN I/O Under PS/8

FORMAT statements each have line numbers and one or more data field specifications. FORMAT statements are used in conjunction with the list of a data transmission statement. Both numeric and alphanumeric field specifications can appear in a FORMAT statement. The FORMAT statement also provides for handling multiple line formats, skipping characters, space insertion, and repetition.

TABLE 7.3

FORMAT Field Specification Codes

Field Type	External Format
E	decimal floating point with E exponents: .324E+10
F	decimal floating point with no exponent: 283.75
I	decimal integer: 79
Aw	alphanumeric data, w characters are to be transmitted. Number of characters transmitted is limited by the number of characters which can be stored in the space allotted for the variable. This maximum number depends on variable type: 6 characters for a real variable and 2 characters

TABLE 7.3 (Cont'd)

FORMAT Field Specification Codes

Field Type	External Format

for an integer variable. The characters are stored in stripped ASCII format. If not enough data is supplied as input to the variables, the data is padded with blanks on the right. For example:

```
      READ (1,2Ø)(M(I),I=1,8)
  2Ø  FORMAT (8A1)
```

if the user types at that point:

```
      123ABC
```

the following are the octal values of M(I):

```
      M(1)=6140    or    1 blank
      M(2)=6240    or    2 blank
        .
        .
        .
      M(6)=0340    or    C blank
      M(7)=4040    or    blank blank
      M(8)=4040    or    blank blank
```

As a second example:

```
      READ(1,2Ø)ALPHA
  1Ø  FORMAT (A6)
```

the user types:

```
      123AB
```

and the octal value of ALPHA is

```
      6162    6301    0240
```

nH — Alphanumeric data can be transmitted directly from the FORMAT statement using Hollerith conversion. H-conversion is referenced by WRITE statements only. The string is specified by the form nHstring where n is the number of characters in the string, including blanks. For example:

```
      2ØØ  FORMAT (17H PROGRAM COMPLETE)
```

will print PROGRAM COMPLETE on the output listing. Alphanumeric fields can be placed among other fields in a FORMAT statement. For example:

```
      21Ø  FORMAT (I5,7H FORCE=F1Ø.5)
```

can be used to output the line:

```
       22 FORCE= 17.689Ø1
```

nX — Blanks can be introduced into an output record or characters skipped on an input record by use of the nX specification. The number n indicates

TABLE 7.3 (Cont'd)

FORMAT Field Specification Codes

Field Type	External Format

the number of blanks or characters skipped and must be greater than zero. For example:

```
5Ø  FORMAT (5H STEPI5,1ØX2HY=F7.3)
```

can be used to output the line:

```
STEP    28     Y=   3.872
```

Output formats are specified in the forms:

```
Ew.d
Fw.d
Iw
Aw
```

where A, E, F, and I designate the conversion type, w is an integer specifying the field width, and d is an integer specifying the number of decimal places to the right of the decimal point. For E and F input, the position of the decimal point in the external field takes precedence over the value of d (Fw.d or Ew.d). For example:

```
15Ø  FORMAT (I5,F1Ø,2,E16.8)
```

could be used to output the line

```
32   -17.6Ø    .59625476E+Ø3
```

on the output listing. Field width (w) should be large enough to include the decimal point, sign, and exponent. Otherwise, the number is right justified in the permissible field, excess digits on the left being lost.

Repetition of a field specification can be indicated by preceding the control character E, F, I, or A by an unsigned integer giving the number of repetitions required. For example:

```
3Ø  FORMAT (2E12.4,3I5)
```

is equivalent to:

```
3Ø  FORMAT (E12.4,E12.4,I5,I5,I5)
```

Repetition of groups is indicated by enclosing the group in parentheses and preceding the whole with the repetition number. For example:

```
4Ø  FORMAT (2I8,2(E15.5,2F8.3))
```

is equivalent to:

```
4Ø  FORMAT (2I8,E15.5,2F8.3,E15.5,2F8.3)
```

Multiple record (line) formats are accomplished by use of the slash character. Where the slash is used in place of a comma, the output device skips to the next line (or card, or whatever the unit record may be). For example:

```
5Ø  FORMAT (3I8/I5,2F8.4)
```

is equivalent to:

```
    FORMAT (3I8)
```

and

```
    FORMAT (I5,2F8.4)
```

In general, numeric input conversion is compatible with most other FORTRAN processors. A few exceptions are listed below:

a. Blanks are ignored except to determine in which field digits fall. Thus numbers are treated as if they were right justified within a field. In an F5.2 format, the following:

```
bbb12
12bbb
.12bb
00012
```

would each be read as the number 0.12 (where "b" represents a blank space).

b. A null line delimited by two CR/LF's is treated as a line of blanks, and blanks are appended to the right of a line (if necessary) to fill out a FORMAT statement. Thus

```
12(CR/LF)
12bbb
bbb12
```

are all identical under an F5.2 format. If an entire line is blank, numeric data from that line is read as all zeroes.

c. No distinction is made between E and F format on input. Thus:

```
100.bb
100E2b
1.E2bb
b10000
```

are all read identically under either an F5.2 or E5.2 format.

7.1.5 FORTRAN Data Transmission Statements

The two FORTRAN data transmission statements are READ and WRITE. Data transmission statements accomplish the I/O transfer of data listed in a FORMAT statement. The two statements are of the form:

```
READ (unit, format) I/O list
WRITE (unit, format) I/O list
```

The input/output lists are lists specifying the order of transmission of the variable values. An element in an input/output list can take one of the following forms:

a. Arithmetic expression: expressions more complicated than a single variable (which can be subscripted) are meaningless in an input operation.

b. The name of an array (1 or 2 dimensional): this indicates that every element of the array is to be transmitted. Elements are transmitted in the order in which they are stored in core. For example:

```
DIMENSION A(2,2)
READ (1,1ØØ)A
```

reads:

```
A(1,1),A(2,1),A(1,2),A(2,2)
```

c. Implied DO loops: of the form:

$$(s_1,s_2,\ldots,s_n,i=m_1,m_2,m_3)$$

repeat the list elements (s_n) with the value of i being equal to m_1 through m_2 having an optional step value of m_3. The m's are integer constants or variables, i is an integer variable, and s_n

are the input/output list elements (possibly including an implied DO loop). For example:

```
DIMENSION A(3,6)
WRITE (1,1ØØ)  I,(A(J,I),J=1,3)
```

will output the values:

```
I,A(1,I),A(2,I),A(3,1)
```

It is important to remember that when using implied DO loops, the entire implied DO loop must be on the same input line or card. An implied DO loop cannot be continued onto the next line with a continuation character.

The READ statement specifies a transfer of information from a selected input device to core memory. The READ statement assumes the following form:

```
READ (d,f) list
```

where d is a device designation which can be an integer constant or an integer variable, f is a FORMAT statement line number, and list is a list of the variables whose values are to be input. The data read by the system is converted to internal form as specified in the referenced FORMAT statement. For example:

```
READ (1,15) ETA,PI,(A(I),I=1,N)
```

The WRITE statement is used to transmit information from core memory to a specified output device. The WRITE statement assumes one of the following forms:

```
WRITE (d,f) list
WRITE (d,f)
```

where d is a device designation (integer constant or integer variable), f is a FORMAT statement line number, and list is a list of variables.

The first form of the WRITE statement causes the values of the variables in the list to be read from memory and written on the device designated in ASCII form. The data is converted to external form as specified by the designated FORMAT statement.

The second form of the WRITE statement causes information to be read directly from the specified FORMAT statement and written on the device designated (Hollerith-type information, generally).

The I/O device designations used in the READ and WRITE statements are as follows:

TABLE 7.4

Device Codes

Device Code	Input Designation	Output Designation
1	Teletype keyboard or Low-speed reader	Teleprinter
2	High-speed reader	High-speed punch
3	Card reader (CR8/I)	Line printer (LPØ8)
4	Assignable device (see Section 7.1.6)	Assignable device (see Section 7.1.6)

Device code 3 is assigned to the card reader (for all READ statements), and the line printer (for all WRITE statements). The card reader uses a two-page device handler, which is too large to be used with the device independent I/O feature (Device code 4). Therefore, the card reader has its own device code.

The line printer is a separate output device because it can require special formatting, such as inserting a Form Feed to skip to the top of a page. The contents of the first column of any line is a control character. These control characters are never printed. They are as follows:

Character in Column 1	Resulting Spacing
space	single space
Ø	double space
1	skip to top of next page (Form Feed)

7.1.6 Device Independent I/O and Chaining

PS/8 FORTRAN provides for device independent, file-oriented, formatted I/O through use of the device number 4 in the READ and WRITE statements and several utility subroutines.

The user must indicate the /I option in order to use device independent input, the /O option to use device independent output. Both options must be indicated for device independent input and output.

NOTE

The card reader cannot be used as an assignable device in device independent I/O statements. The run time FORTRAN system will not permit any two page device handlers at this time.

7.1.6.1 The IOPEN Subroutine

The subroutine IOPEN prepares the system to accept input from a specified device when device code 4 is used in a READ statement. IOPEN takes two arguments which are interpreted as Hollerith strings. After a

```
CALL IOPEN (A,B)
```

any READ statement reading from device 4 will read from the file specified by B (which must have the extension .DA) on the device specified by A. For example:

```
CALL IOPEN ('DTA5','INPUT')
```

will prepare for input from the file DTA5:INPUT.DA

```
CALL IOPEN ('F1',Ø)
```

will prepare for input from the device F1.

If the filename and device name are read with READ statements using A format, the two names must be read using A6 format and the remaining characters filled with @ signs, not blanks.

7.1.6.2 The OOPEN Subroutine

The subroutine OOPEN prepares the system to send output to a specified device when device code 4 is used in a WRITE statement. The arguments of OOPEN are treated like those of IOPEN. Future WRITE statements using device 4 write on the device and file specified in the call to OOPEN. An error message is printed if the program has previously issued a CALL OOPEN without issuing a subsequent CALL OCLOSE. For example:

```
CALL OOPEN ('PTP',Ø)
```

prepares device 4 to output on device PTP:

```
CALL OOPEN ('SYS','LADE')
```

prepares device 4 to output to the file SYS:LADE.DA.

7.1.6.3 The OCLOSE Subroutine

The subroutine OCLOSE is called with no arguments. Its function is to terminate output on the output file opened by OOPEN. If OCLOSE is not called, the output file will never exist on the specified device.

7.1.6.4 The CHAIN Subroutine

A call to the subroutine CHAIN terminates execution of the calling program and starts execution of the core image on the system device as specified by the argument to CHAIN. Variables in COMMON storage are not disturbed. For example:

```
CALL CHAIN('PROG')
```

causes the file SYS:PROG2.SV to be loaded and started. Notice that PROG2 must be compiled and stored on the system device in order to be successfully accessed.

7.1.6.5 The EXIT Subroutine

To return to the Keyboard Monitor from a FORTRAN program, the EXIT subroutine is used, as follows:

```
        CALL EXIT
```

7.1.6.6 FORTRAN Data Files

When doing FORTRAN output onto DECtape or disk into a file which is to be read only as a data file by another FORTRAN program, a significant time saving can be obtained by using A6 format to output floating-point variables and A2 format to output integer values. The same format specifications must be used when the data is read. The data file is not an ASCII file and should not be edited with EDIT. The file should only be moved by PIP in image mode (/I option).

7.1.7 Mixing SABR and FORTRAN Statements

An S in column 1 of an input line identifies that line as containing SABR code. This feature is very useful for doing things which are undefined in the FORTRAN language. For example:

```
      DIMENSION M(1Ø)
        .
        .
        .
      J=M(1)
      DO 55 K=2,1Ø
      L=M(K)
S     TAD    \L
S     AND    \J
S     DCA    \J
55    CONTINUE
```

This section of code will form the logical AND of M(1) through M(1Ø) in the variable J.

Notice that whenever a FORTRAN variable is used in a SABR statement, the variable name is preceded by a backslash (\). FORTRAN line numbers referenced in SABR statements are also preceded by a backslash for identification purposes. (A backslash is produced by typing a SHIFT/L.)

7.1.8 8K FORTRAN Statement Summary

TABLE 7.5

8K FORTRAN Language Summary

Statement	Definition
Arithmetic Statements	
v=e	v is a variable (scalar or array); e is an expression.
Control Statements	
GOTO n	Transfer control to the statement numbered n.
GOTO $(n_1,n_2,\ldots,n_i)j$	Where n_i are statement numbers and j is a scalar integer variable. This statement transfers control to the j^{th} member of the series of n_i.
IF (expression) n_1,n_2,n_3	This statement transfers control to the statement numbered n_1, n_2, or n_3 if the value of the numeric expression is less than, equal to, or greater than zero, respectively. The expression can be simple or complex.
DO n $i=m_1,m_2,m_3$	Repeat execution through statement n, beginning with $i=m_1$, incrementing by m_3, while i is less than or equal to m_2. If m_3 is omitted, it is assumed to be 1. m's and i's cannot be subscripted. m's can be either integer numbers or integer variables. i is an integer variable.
CONTINUE	Dummy statement, used primarily as a target for transfers, particularly as the last statement in the range of a DO loop. A DO loop need not end with a CONTINUE statement.
PAUSE PAUSE n	Temporarily suspend execution. The octal equivalent of the decimal number n is displayed in the accumulator. Program execution can be resumed by depressing the CONT key on the console.
STOP	Terminate execution.
END	Terminate compilation; must be the last statement in a program.

TABLE 7.5 (Cont'd)

8K FORTRAN Language Summary

Input/Output Statements

FORMAT(s_1,s_2,...,s_n)	Where s_n are data field specifications, this statement is used with either a READ or WRITE statement. See Section 7.1.4 for details.
READ (u,f) list	Where u is a device designation (integer constant or integer variable); f is a FORMAT statement number; and list is a list of variables.
WRITE (u,f) list	Where u is a device designation (integer constant or integer variable), f is a format reference, and list is a list of variables.

Specification Statements

COMMON v_1,v_2,...,v_n	Specified variables or arrays are stored in an area available to other programs.
DIMENSION a_1,a_2,...,a_n	Used to declare variable names to be array names and specify the number and bounds of each one and two dimensioned array.
EQUIVALENCE (v_1,v_2,...), (v_i,v_{i+1},...)	The inclusion of two or more variable or array names in a parenthetical list indicates that the quantities in the list are to share the same memory location and hence have the same value. Subscripts of array variables must be integer constants. Names must not appear in both EQUIVALENCE and COMMON statements.

Subprogram Statements

FUNCTION v(a_1,a_2,...,a_n)	Declares the program which follows to be a function subprogram. v is the name of the function being defined. v must appear as a scalar variable and be assigned a value during execution of the subprogram. See Chapter 15, Programming Languages 1970 for a detailed explanation.

TABLE 7.5 (Cont'd)

8K FORTRAN Language Summary

SUBROUTINE $v(a_1,a_2,\ldots,a_n)$	Declares the program which follows to be a subroutine subprogram. The arguments in the list(s) are dummy arguments representing the arguments of the subprogram. Dummy arguments must agree in number, order, and type with the arguments used by the calling program. See Chapter 15, Programming Languages for a detailed explanation.

CALL v
CALL $v(a_1,a_2,\ldots,a_n)$
	Statement used to transfer control to a subroutine subprogram. v is the subroutine name in the SUBROUTINE statement. The arguments can be of any type, but must agree in number, order, type and array size with the arguments in the SUBROUTINE statement. One or more of the arguments can be used to return results to the calling program. For example:
	`CALL EXIT`
	`CALL TEXT(VALUE,123,275)`
	`CALL TECK('MAX',3)`

RETURN	Returns control from a subprogram to the calling program. Each subprogram must contain at least one RETURN statement. RETURN cannot be used in the main program.

7.1.9 FORTRAN Error Messages

FORTRAN Compiler error messages are self-explanatory.

```
ARITHMETIC EXPRESSION TOO COMPLEX
EXCESSIVE SUBSCRIPTS
ILLEGAL ARITHMETIC EXPRESSION
ILLEGAL CONSTANT
ILLEGAL CONTINUATION
ILLEGAL EQUIVALENCING
ILLEGAL OR EXCESSIVE DO NESTING
ILLEGAL STATEMENT
ILLEGAL STATEMENT NUMBER
ILLEGAL VARIABLE
MIXED MODE EXPRESSION
SYMBOL TABLE EXCEEDED
SYNTAX ERROR (usually indicates illegal punctuation)
```

If an error is discovered in the user's FORTRAN program, the Compiler prints the incorrect line, followed by an error message. Although Compiler output will be suppressed, the rest of the user's program is read, and additional error messages are printed where necessary.

The following error messages have been added to the PS/8 version of FORTRAN:

Message	Explanation
I/O	A device handler has signalled an I/O error
NO ROOM FOR OUTPUT	The file FORTRN.TM cannot fit on the system device.
SABR.SV NOT FOUND	The SABR Assembler is not present on the system device.
NO END STATEMENT	The input to the Compiler has been exhausted.
COMPILER MALFUNCTION	The meaning of this message has been extended to cover various unlikely monitor errors.

During execution, the various library programs check for certain errors and print error messages in the form:

```
XXXX  ERROR AT LOC NNNNN
```

where XXXX is the error code and NNNNN is the location of the error.

TABLE 7.6

FORTRAN Library Error Messages

Error Code	Meaning
	The following errors are fatal and cause a return to the Keyboard Monitor.
ALOG	Attempt to compute log of negative number.
IOER	One of the following has occurred: 1) Device independent input or output attempted without /I or /O options, 2) Bad arguments to IOPEN or OOPEN, or 3) Transmission error while doing I/O.
CHER	File specified as argument to CHAIN not found on system device.
FMT1	Invalid Format Statement
	The following input errors are fatal unless input is coming from the Teletype, in which case the entire READ statement is tried again.
FMT2	Illegal character in I format.
FMT3	Illegal character in F or E format
	The following errors do not terminate execution of the user's program.
DIVZ	Division by zero - very large number is returned.
EXP	Argument to EXP too large - very large number is returned.
OVFL	Floating point overflow - very large number is returned.
FLPW	Negative number raised to floating point power - absolute value taken.
SQRT	Attempt to take square root of negative number - absolute value used.
FIX	Attempt to fix a number >2047; 2047 is returned.

In addition, the error message

USER ERROR 1 AT XXXX

means that the user tried to reference an entry point of a program which was not loaded. XXXX has no meaning.

To pinpoint the location of a library program execution error:

a. Determine, from the storage map, the next lowest numbered location (external symbol) which is the entry point of the program or subprogram containing the error.

b. Subtract, in octal, the entry point location of the program or subprogram containing the error from the location of the error indicated in the error message.

c. From the assembly symbol table, determine the relative address of the external symbol found in step a and add that relative address to the result of step b.

d. The sum of step c is the relative address of the error, which can then be compared with the relative addresses of the numbered statements in the program.

7.1.10 Implementation Notes

7.1.10.1 Alphanumeric Data Within FORMAT Statements

Alphanumeric data can be transmitted directly from the FORMAT statement by two different methods: H-conversion or the use of single quotes.

Hollerith (H) format is used in WRITE statements only. An attempt to use H format specifications with a READ statement will cause characters from the format field to be either printed or punched. This can occasionally be a useful feature, since it provides a simple way of identifying data that is to be read from the keyboard. For example, the following instructions:

```
      READ (1,3Ø)A,B
   3Ø FORMAT (4HA = ,F7.2/4H = ,F7.2)
```

would cause A= and B= to be printed before the data is read.

The same effect is achieved by merely enclosing the alphanumeric data in single quotes. The result is the same as in H-conversion; on output the characters between the single quotes (including blanks) are written as part of the output data. For example, when referenced from a WRITE statement,

```
FORMAT ('PROGRAM COMPLETE')
```

would cause PROGRAM COMPLETE to be printed. This method eliminates the need to count characters.

7.1.10.2 Subscripting

Since excessive subscripting tends to use core memory inefficiently, it is suggested that subscripted variables be used judiciously. For example, the statement

```
A=((B(I)+C2)*B(I)+C1)*B(I)
```

could be rewritten with a considerable saving of core memory as follows:

```
T=B(I)
A=((T+C2)*T+C1)*T
```

CAUTION

Programs containing subscripting compiled with the new FORTRAN compiler (on or after October 1970) will not run with a FORTRAN library previous to October 1970. A new subscripting algorithm has been used with the new FORTRAN compiler which occupies 50% less core space and allows the user to run an up to 35% larger program depending on the amount of subscripting used than with the earlier 8K FORTRAN. Similarly, programs containing subscripting developed with the new FORTRAN compiler will not run under the old FORTRAN system.

7.1.10.3 DO Loops

DO loops are treated slightly differently in 8K FORTRAN than in most compilers. The index is tested before the range of the DO is executed. Therefore, in the following example

```
      DO  2Ø N =1,M
          .
          .
          .
2Ø    CONTINUE
```

the instruction between the DO statement and statement 20 is never executed if M is less than one.

7.1.10.4 PAUSE Statement

The PAUSE statement may be used for a variety of reasons to temporarily suspend program execution. In some cases, the PAUSE statement can be used to give the operator a chance to change data tapes or to remove a tape from the punch. When this is done it is necessary to follow the PAUSE statement with a call to the OPEN subroutine. This subroutine initializes the I/O devices and sets hardware flags that may have been cleared by pressing the tape feed buttons. Example:

```
      PAUSE
      CALL IOPEN
```

7.1.10.5 EQUIVALENCE Statement

Because of core memory restrictions within the compiler, variables can not appear in EQUIVALENCE statements more than once. Thus,

```
      EQUIVALENCE (A,B,C)
```

would be valid, but the statement

```
      EQUIVALENCE (A,B),(B,C)
```

would not compile correctly.

7.1.10.6 Size of FORTRAN Programs

The maximum size of any FORTRAN program is 36 octal or 30 decimal pages of code.

PS/8 can run FORTRAN programs in 8 to 32K of core. No one program or subprogram can be longer than 4K, however.

The user can estimate the size of his program as follows: Take the amount of core available on the system (at least 8K) and from it subtract 4K for the linkage subroutines; external symbol table; and I/O, math, error, and utility subroutines. From the remainder subtract the amount of storage required for data. The remaining space can be used to hold FORTRAN coding, at the rate of 50-70 FORTRAN statements per 1K of core.

One way to have a longer FORTRAN program in core than is usually possible is to divide a FORTRAN program into three chained segments:

Segment 1 - inputs data into COMMON storage
Segment 2 - FORTRAN program for data processing
Segment 3 - does output to desired device(s)

When this technique is used the I/O handler routines are not loaded with the second program. The second program can then be longer than if those routines were in core while the processing is being done.

When chaining to a subroutine, the user must be sure he has compiled, loaded, and saved a complete runnable main program on the system device. This program is brought into core by the FORTRAN CHAIN subroutine.

7.1.10.7 Using FORTRAN or SABR with the Interrupt ON

SABR code can be run with the interrupt on, providing the user supplies his own interrupt handling code. That code which is executed when the interrupt is off must not call any of the SABR subroutines and must be independent of all SABR or library subroutines and linkage subroutines.

With the interrupt on the user should not call exit routines or do any generalized (device-independent) I/O, unless those routines are modified to make allowances for interrupts.

7.1.10.8 Using PAL-8 with SABR or FORTRAN

It is possible to call PAL-8 subroutines from a SABR or FORTRAN program. The user should build a core image of the

running program and return to the Keyboard Monitor by typing $ (ALT MODE key) on the last Linking Loader Command. He should then save the core image. The core image file (.SV) can be used as input to the absolute loader (ABSLDR) with the /I option, followed by the binary of the PAL-8 routine. For example:

```
.R ABSLDR
*DTA7:CHAIN2.SV /I
*PALSUB.BN /G$
```

The above calls the Absolute Loader, loads the core image CHAIN2.SV and then merges the PALSUB.BN program with it. Execution starts at location 200 and, when completed, the system returns to the Keyboard Monitor for further instructions.

7.1.10.9 Errors

Undefined statement numbers are not detected until the assembly phase. A U error message is given. See list of SABR error messages.

7.2 8K SABR ASSEMBLER

The 8K SABR assembler can be used as the automatic second pass of the FORTRAN Compiler, called separately to do assemblies of FORTRAN-compiled files, or used as an independent assembler with its own assembly language. As explained in Section 7.1.7, it is also possible to mix SABR statements into a FORTRAN program to expand the capabilities of the FORTRAN language.

The 8K SABR assembler is described in Chapter 14 of Programming Languages. The description in this section summarizes the SABR language and the capabilities of the assembler. For more complete details and examples of usage, the reader is referred to Programming Languages.

7.2.1 Calling and Using 8K SABR

Unless otherwise specified, the SABR assembler is called automatically by the system to assemble the output of a FORTRAN compilation. At other times the user can call SABR by typing:

```
.R SABR
```

where the dot was printed by the Keyboard Monitor. When the Command Decoder prints a star at the left margin the user types the appropriate I/O files and any of the acceptable options.

The line to the Command Decoder consists of 0 to 3 output files (the first for binary output, the second for the listing, and the third for Linking Loader loading map output). 1 to 9 input files are then indicated. A null binary file is assumed to be SYS:FORTRL.TM if the /L or /G options are specified. Otherwise, a null binary output file indicates that no binary output is to be generated.

The assumed extensions for SABR are as follows:

File Type	Extension
input	.SB
binary output	.RL
listing output	.LS

Table 7.7 describes the options which can be included in a command string to 8K SABR.

TABLE 7.7
SABR I/O Options

Option	Meaning
/L	Call the Linking Loader at the end of the assembly and load the specified binary file. If a binary output file is not specified, then the temporary file FORTRL.TM is loaded into core and deleted from the file device. The Loader then either returns to the Keyboard Monitor with a core image in core or asks for more input, depending on whether an ALT MODE or RETURN key terminated the input line.
/G	Call the Linking Loader, load the program into core and begin execution. If a binary output file is not specified, then FORTRL.TM is loaded into core and deleted from the file device. If a starting address is not specified (using the options to the Linking Loader), control is sent to the program entry point MAIN (FORTRAN Compiler gives this name automatically to the main program.
/F	Indicates that the input file is an 8K FORTRAN output file.
/N	Output the symbol table but not the rest of the listing (applicable only if a listing file is specified).
/S	Omit the symbol table from the listing (applicable only if a listing file is specified).

When the /L or /G options are specified, any options to the Linking Loader (described in Section 7.3.1) can be included in the command string for SABR. This does not include the /L (Library) option of the Linking Loader, since it would conflict with the SABR /L option.

NOTE

The FORTRAN Compiler automatically generates an entry point named MAIN whose address is the beginning of the program. When writing a main program in SABR, the user should specify the entry point MAIN with the entry pseudo-op in order to symbolically specify the starting address to the Linking Loader (otherwise the starting address must be specified to the Loader as a five digit address).

7.2.2 Examples of 8K SABR I/O Specification Commands

Example 1:

```
.R SABR
*FORTRN.TM/F/G
```

DSK:FORTRN.TM is assembled as a FORTRAN output file and the relocatable binary is loaded and started at the entry point MAIN.

Example 2:

```
.R SABR
*SYS:TEERL,TTY:<TEE /S
```

The input file TEE.SB (or TEE) on DSK: is assembled. The relocatable binary goes to the output file TEERL.RL on SYS:, the listing without a symbol table goes to the Teletype.

7.2.3 SABR Statements

SABR symbolic code is written as a sequence of statements and is usually prepared on-line through the Symbolic Editor program. SABR statements are virtually format free. A statement line is composed as follows:

```
label, operator  operand  /comment
```

each element of which is separated from the others by spaces or tabe. Labels require a command following them and comments must be preceded by a slash. SABR generates one or more machine (binary) instructions or data words for each source statement.

An input line can be up to 72 characters long. The RETURN key is both a statement and a line terminator. The semicolon can be used to terminate an instruction without terminating the line, permitting the user to have more than one instruction per line of listing. Null lines created by typing an extra RETURN key appear as blank lines in the program listing.

Table 7.8 contains a list of all special characters used in the SABR language.

TABLE 7.8

Special SABR Characters

Character	Usage
,	delimits a symbolic address label
/	indicates the start of a comment
(	indicates a literal
	(D indicates that the numeric literal is decimal
	(D32 is equivalent to octal 40
	(K indicates numeric literal is octal
	(K-32 is equivalent to 7746 octal
"	precedes an ASCII constant, for example:
	"; is equivalent to 273 octal
	-"A is equivalent to -301 octal
-	negates a constant
#	increases the value of the preceding symbol by one. For example: TAD LOC# is equivalent to the PAL statement TAD LOC+1.
RETURN key	terminates a statement line
;	terminates an instruction
space	separates and delimits items on the statement line
TAB	same as space

LINE FEED, FORM FEED, and RUBOUT are ignored. All other characters are illegal except when used as ASCII constants following a quote or in comments or text strings.

7.2.4 Statement Elements

Statements are composed of labels, operators, operands, and (optionally) comments. A label is a symbolic name or location tag created by the programmer to symbolically identify the address of a statement in the program. Subsequent references to the statement can be made by referencing the label. For example:

```
SAVE, Ø
ABC,  TAD SAVE
```

SAVE and ABC are labels.

An operator can be one of the following:

a. a memory reference instruction (direct or indirect)
b. an operate or IOT microinstruction
c. or a pseudo-operator

An operand can be a user-defined address symbol, a literal, or a numeric constant.

7.2.5 Symbols

Permanent symbols are predefined and maintained in SABR's permanent symbol table (see Appendix C). Additional permanent symbols are defined using the OPDEF and SKPDF pseudo-operator. User-defined symbols are 1 to 6 alphanumeric characters long and conform to these rules:

a. Characters must be alphanumeric, A-Z, 0-9
b. First character must be alphabetic
c. Only the first six characters are significant
d. Cannot be the same as any permanent symbol
e. Must be defined only once. A symbol is defined when it appears as a symbolic address label or in an ABSYM, COMMN, OPDEF, or SKPDF statement.
f. No more than 64 different user-defined symbols on any one core page.

When an address label appears alone on a line (with no instruction or parameter) the label is assigned the value of the next address assembled. For example:

```
TAG1,
TAG2,  3Ø
TAG3,
```

TAG1 and TAG2 are equivalent symbols and are each assigned the value 30. TAG3 is defined as being the octal location value of TAG2, plus 1.

7.2.6 Constants

Numeric constants consist of strings of from 1 to 4 digits, optionally preceded by a + or - sign. The digit string is inter-

preted as either octal or decimal according to the latest permanent mode setting by an OCTAL or DECIM pseudo-operator. Octal mode is assumed at the beginning of an assembly. Digits 8 and 9 must not appear in an octal string.

ASCII constants are also allowed as shown in Table 7.8. ASCII constants are preceded by a " character.

See also Table 7.8 for a way to increment operands using the # character. The # feature is intended primarily for manipulating dummy variables when picking up arguments from external subroutines and returning from external subroutines.

7.2.7 Pseudo-Operators

Table 7.9 contains a list of the pseudo-operators used with 8K SABR and a brief description of their use. For a detailed description of each pseudo-op, see Programming Languages.

TABLE 7.9

8K SABR Pseudo-Operators

Mnemonic Code	Operation
ABSYM	Direct absolute symbol definition, used to indicate an absolute core address. For example: `ABSYM TEM 177  /PAGE ZERO ADDRESS`

ARG	Argument for subroutine call, indicating a value to be transmitted, one value per ARG statement. Used only with CALL. For example:
	
	`N1, ARG (5Ø`
	`N2, ARG LOCATN`

BLOCK	Reserve Storage block, reserves n words of core by placing zeros in them. For example:
	
	`BLOCK 2ØØ  /RESERVE 3ØØ`
	`BLOCK 1ØØ  /(OCTAL) LOCATIONS`

CALL	Call external subroutine. For example: `CALL 2,SUBR` where 2 is the number of arguments to be passed and SUBR is the subroutine name.

TABLE 7.9 (Cont'd)

8K SABR Pseudo-Operators

Mnemonic Code	Operation
COMMON	Common storage definition, used to name locations in field 1 as externals to be referenced by any program. For example: `A, COMMN 2Ø  /2Ø WORDS IN COMMON`
CPAGE	Check if page will hold data, followed by the number of words of code which must be kept together in a unit on a page. That number of words following the CPAGE will be assembled as a unit on the next available core page.
DECIM	Decimal conversion, numeric conversion interprets all numbers input as being decimal numbers.

DUMMY	Dummy argument definition, used in passing arguments to and from subroutines. DUMMY variables are defined in the subprograms which reference them. For example:
	
	`ENTRY A1`
	`DUMMY X`
	`DUMMY Y`

EAP	Enter automatic paging mode, restore automatic paging (see LAP).
END	End of program or subprogram.

ENTRY	Define program entry point, used at beginning of subprograms to give name of entry point for the Linking Loader. For example:
	
	`ENTRY  SUBROU`
	`SUBROU,  BLOCK 2`

FORTR	Assemble FORTRAN tape.
I	Symbolic representation for indirect addressing. For example: `DCA  I  ADD`
IF	Conditional assembly, of form: `IF NAME,  7` if the symbol NAME has been previously defined, the statement has no effect. If NAME is not defined, the next 7 symbolic instructions are NOT assembled.

TABLE 7.9 (Cont'd)

8K SABR Pseudo-Operators

Mnemonic Code	Operation
LAP	Leave automatic paging. Assembler is initially set for automatic jumps to the next core page when the current page is full (or upon REORG or PAGE statements). This feature can be suppressed with LAP.
OCTAL	Octal conversion, numeric conversion is originally set to octal and can be changed back to octal after a DECIM pseudo-op has been used.
OPDEF	Define non-skip operator. For example: OPDEF DTRA 6761
PAGE	Terminate current page, begin assembly of succeeding instructions on next core page.
PAUSE	Pause for next tape, designed to allow large source tapes to be broken into several smaller segments. Assembly is continued by pressing the CONT switch.
REORG	Terminate page and reset origin; origin settings are always to the first address of a page. For example: REORG 1ØØØ
RETRN	Return from external subroutine, the name of the subroutine being left must be specified. Before the RETRN statement is used, the pointer in the second word of the subprogram entry must be incremented to the point following all arguments in the calling program (after the CALL statement).
SKPDF	Define skip-type operator. For example: SKPDF DTSF 6771
TEXT	Text string, similar to BLOCK, except that the argument is a text string. Characters are stored in six-bit stripped ASCII with a printing character used to delimit the string. For example: TAG, TEXT /123*/ the string would be stored as: 6162 6352 0000
ACH ACM ACL	The floating-point accumulator (in field 1).

7.2.8 SABR Operating Characteristics

SABR assembles programs page-by-page, building various tables as instructions are read. Literals and off-page pointers are gathered together in a table at the end of each program page. The LAP and EAP pseudo-ops can be used to control the automatic paging facilities of SABR.

7.2.9 Symbol Table

One and two character symbols require three symbol table words. Three and four character symbols require four words, and five and six character symbols, five words.

The symbol table, not counting permanent symbols, contains about 1800 (decimal) words of storage. This space is also used when there are unresolved forward and external references, temporarily stored as two-word entries.

Symbols are listed in alphabetic order at the end of the assembly pass 1 with their relative addresses beside them. The following flags are added to denote special types of symbols:

TABLE 7.10

SABR Symbol Flags

Flag	Meaning
ABS	Address referenced by this symbol is absolute.
COM	Address is in COMMON.
EXT	Symbol is external, and may or may not be defined in this program. If not defined in this program, it is assumed to be defined in another program.
OP	Symbol is an operator.
UNDF	Symbol is not external and has not been defined in the program. Programmer error.

7.2.10 SABR Error Messages

Because SABR is a one-pass, automatic paging assembler, object errors are difficult to correct. If there are errors in

the source, the assembled binary code will be virtually useless.

During assembly, error messages are printed at the Teletype as they occur in the form:

```
C  AT  LOC  +0004
```

which means that an error of type C has occurred at the fourth instruction after the location tag LOC. The line count includes comment lines and blank lines. The following messages can occur:

TABLE 7.11

SABR Error Codes

Error Code	Meaning
A	Too many or too few ARG statements follow a CALL statement.
C	An illegal character appears on the line.
D	A device handler has returned a fatal error condition.
L	/L or /G option was indicated, but the LOADER.SV file does not exist on the system device.
M	A symbol is multiply defined. Listings of programs with multiple definitions have unmarked errors.
I	An illegal syntax has been used, one of the following: a pseudo-op with improper arguments a quote mark with no argument a non-terminated text-string an improper address an illegal combination of micro-instructions
E	There is no END statement.
S	Either the symbol table has overflowed, common storage has been exhausted, more than 64 different user-defined symbols occurred in a core page, or more than 64 external symbols have been declared. Could also indicate a system error such as overflowed output file.
U	No symbol table is being produced, but there is at least one undefined symbol in the program.
UNDF	Undefined symbol, printed in the symbol table listing.

7.3 LINKING LOADER

The Linking Loader can be called automatically to load or load and start a FORTRAN or SABR program. The Linking Loader can also be called independently to load or load and start a relocatable binary file stored on a device.

The Linking Loader is capable of loading and linking a user's program and subprograms in any field(s) of memory. It is even capable of loading programs over itself. The Linking Loader has options which can obtain storage map listings of core availability for the user.

The Linking Loader has the capability to search program libraries for subroutines which are referenced by the program in core and load those subroutines needed. A library is a collection of relocatable subroutines (FORTRAN or SABR output) with a directory at the beginning to facilitate searching. Any library can be searched by using the /L option to the Loader, but the system library, LIB8.RL, is searched automatically just before the Loader completes the building of a core image of the user's program. If LIB8.RL is not on the system device, there is no automatic library search. A program which will allow the user to build his own subroutine library will be available early in 1971.

The Linking Loader is capable of loading any number of user and library programs into any field of memory. Several programs are usually loaded into each field. Because of the space reserved for the Linkage Routines, the available space in field 0 is three pages smaller than in all other fields.

Any common storage reserved by the programs being loaded is allocated in field 1 from location 200 upwards. The space reserved for common storage is subtracted from the available loading area in field 1. The program reserving the largest amount of common storage must be loaded first.

The Run-Time Linkage Routines necessary to execute SABR programs are automatically loaded into the required areas of every field by the Linking Loader as part of its initialization. The user needs to know nothing more about these routines than the particular areas of core they occupy (see Section 7.3.3).

7.3.1 Calling and Using the Linking Loader

The user can automatically call the Linking Loader following a SABR assembly (of either a SABR program or a SABR-assembled FORTRAN program) by use of the /L and /G options. For details on automatic calling of the Linking Loader, see Sections 7.1.1 and 7.2.1.

Where the user wishes to call the Linking Loader specifically to load or load and start a relocatable binary file, he issues the command:

```
.R LOADER
```

to the Keyboard Monitor (which printed the dot). The Command Decoder replies by printing a star at the left margin and the user then indicates input/output files and any desired options.

There can be 0 to 1 output files, and 1 to 9 input files. Only one binary program per file is permitted. The assumed extension for input files is .RL. The output file, if indicated, is used to hold a map of the loaded program.

The user has the ability to specify all options and operations to be performed on one line or have various operations performed individually. Where all options are being specified at one time, the line to the Command Decoder contains the complete instructions for the Linking Loader. If operations are to be done individually, the user can type a command, entered with the RETURN key, and that command will be done, with another command expected when the first is completed. To indicate the last command the user types an ALT MODE character or ends the last command with a /G option (start the program).

The options to the Linking Loader are as shown in Table 7.12.

TABLE 7.12

Linking Loader I/O Options

Option	Meaning
/I	A program doing device-independent input is to to be loaded. (This feature costs the user 3 pages of core.)
/O	A program doing device-independent output is to be loaded. (This feature costs the user 3 pages of core.) If both /I and /O are indicated, 6 pages of core are used to handle device-independent I/O. /I and /O, if used, must be given before or on the first input line specifying files to be loaded. For example: *INPUT,FILES /O$ is acceptable, but *INPUT * /O FILES is not legal and will generate an error message.
/G	Start the program after processing the rest of the command string. Execution starts at the symbol MAIN unless otherwise indicated.
=n	Specifies the starting address of the program if other than the entry point MAIN; where n is an octal number up to 5 digits long.
/M	Output a map of the loaded programs onto the outfile specified, followed by a count of the free pages in each field. If no output file is specified, the map is put onto the teleprinter. The assumed extension for the map output file is .MP. The map is printed after the rest of the command line is processed.
/U	Like /M, but only outputs undefined symbols.
/P	Like /M, but only outputs counts of free pages in each field.
/n	Where n is an integer in the range 0 to 7, inclusive, search through the available fields starting at field n for space large enough to hold each input file. Only one binary program can be in each input file. If n is not specified, the Loader starts looking at field 0.
/R	Restart loading process (forget all previously loaded programs). This command is equivalent to restarting the Linking Loader, but is much faster for DECtape systems, since no tape motion is involved.

TABLE 7.12 (Cont'd.)

Linking Loader I/O Operations

Option	Meaning
/L	Load the first input file as a library file (Loader expects a Library Directory as the first block of the file). All other input files on the line are ignored.

The Core Availability option (/P), causes the number of free pages of memory in every field of memory to be printed in a list on the teleprinter. For example, if the user has a 16K configuration, a list like the following might be printed:

```
ØØØ2     (number of free pages in field 0)
ØØ1Ø     (number of free pages in field 1)
ØØ3Ø     (number of free pages in field 2)
ØØ36     (number of free pages in field 3)
```

The number of pages initially available in field 0 is 0033 and in all other fields is 0036.

The Storage Map option (/M), when selected, causes a list of all program entry points to be printed along with the actual address at which they have been loaded. Entry points of programs which have been called but which have not been loaded are also listed along with a U flag for "undefined". Such flagged programs must be loaded before execution of the user's programs are possible. The core availability list is automatically appended to the storage map. A sample is shown below for an 8K machine:

```
MAIN    1Ø2ØØ
READ    Ø1Ø55
WRITE   Ø1Ø66
IOH     Ø3Ø31
ERROR   ØØØØØ U
GENIO   ØØØØØ U
FDV     Ø4722
CLEAR   Ø5247
IFAD    Ø5131
FMP     Ø4632
ISTO    Ø5Ø74
STO     Ø4447
```

```
FLOT    Ø521Ø
FAD     Ø4Ø1Ø
DIV     ØØØØØ U
IREM    ØØØØØ U
FSB     Ø4ØØØ
FLOAT   Ø5Ø46
FIX     Ø4513
IFIX    Ø4561
CHS     Ø5231
ØØ11
ØØ33
```

7.3.2 Examples of I/O Command Strings

The user having typed:

```
.R LOADER
```

the system returns a star and expects input to the Command Decoder. The following is an example of input to the Command Decoder:

```
*PROG,DTA2:SUBR1,SUBR2/G
```

which loads DSK:PROG.RL, DTA2:SUB1.RL, DTA2:SUB2.RL, loads any necessary library routines requested, and starts the program at the entry point MAIN. The same process could have been done as follows:

*PROG	load DSK:PROG.RL
*/U . . .	get a list of undefined symbols on the teleprinter (symbols go here)
*DTA2:SUBR1,SUBR2	load DTA2:SUBR1.RL,SUBR2.RL
*LPT:/M<$	put loading map on the line printer load the binary of any library routines requested by the program and exit. ($ is printed by the ALT MODE key.)
.SAVE DTA2 FORTPG	save the core image on DTA2 as FORTPG.SV.
.START	start the core image at its starting address (entry point MAIN in this case).

7.3.3 Linking Loader Error Messages

The Linking Loader gives error messages in the form:

ERROR nnnn

The different values of the nnnn error code are listed in Table 7.13.

TABLE 7.13

Linking Loader Error Messages

Error Code	Meaning
0000	/I or /O specified too late.
0001	Symbol table overflow, more than 64 subprogram names.
0002	Program will not fit into core.
0003	Program with largest common storage area was not loaded first.
0004	Checksum error in input tape.
0005	Illegal relocation code.
0006	An output error has occurred while reading a binary file.
0010	No starting address has been specified and there is no entry point named MAIN.
0011	Input device handler requires two pages and will not fit into core where device-independent I/O is being done.
0012	I/O error on system device.

CHAPTER 8
LOADING AND OPERATING PROCEDURES

8.1 LOADING PS/8 ON A DECTAPE SYSTEM

It is suggested that the user copy the PS/8 System DECtape (DEC-P8-MSUB-UC) onto a certified PDP-8 DECtape, with the DECtape copy program (DEC-08-YPTA-PB), before using the PS/8 System. The copy can be used, and the original stored in a safe place, in case part of the PS/8 system should ever be inadvertently destroyed during subsequent use.

If your system uses DECtape as the system device (SYS and DSK), use the following loading procedures:

1. Mount the PS/8 system DECtape (DEC-P8-MSUB-UC) on DECtape unit 0 (unit 8 on some tape drives).
2. Set the LOCAL/REMOTE switch to REMOTE, and the WRITE LOCK switch to WRITE LOCK.
3. Deposit the following bootstrap routine with the console switches (for instructions on how to do this, see the description of how to load the RIM Loader in either Introduction to Programming or Programming Languages.

Location	Instruction
7613	6774
7614	1222
7615	6766
7616	6771
7617	5216
7620	1223
7621	5215
7622	0600
7623	0220
7754	7577
7755	7577

4. Turn the Teletype control knob to LINE. Now set the switch register to 7613, then press the LOAD ADD and START keys in that order. The DECtape should rock two or three times and the PS/8 Keyboard Monitor should respond with a dot at the left margin.
5. To be sure the Keyboard Monitor is running, type a CTRL/C. The system should respond by printing "↑C" followed by another dot on the next line. Now set DECtape unit 0 to WRITE ENABLED and you are ready to begin programming with PS/8.

If the system is not functioning properly after these steps, either

the bootstrap loader was not deposited correctly, the DECtape has been damaged or not mounted correctly, or a hardware malfunction has occurred. First check the bootstrap loader and try the procedure again. If the system still does not appear to function, contact the local DEC sales office.

8.2 LOADING PS/8 ON A DISK OR DECTAPE SYSTEM FROM PAPER TAPE

If your system uses DF32 disks, an RF08 disk, an RK8 disk, or DECtape, use the following loading procedures:

1. First load the RIM and Binary Loaders. For instructions, see Introduction to Programming or Programming Languages).
2. Load the PS/8 binary tape with the Binary Loader, as follows: Put the tape in the high-speed reader, place 7777 in the switch register (set the Data Field to zero and the Instruction Field to the field in which the Binary Loader is present), press the LOAD ADD key, depress switch register bit zero, and press the START key. The tape will be read and will stop on the trailer code.
3. Several Configuration tapes (CONFIG) are available for different disks. After the PS/8 Binary tape has been read, place the appropriate Configuration binary tape in the high-speed reader and press the CONT key on the console to load this tape. After reading each tape (in steps 2 and 3) check to see that no checksum errors have occurred. If any accumulator lights are on after a tape is read, a checksum error was encountered and that tape should be reloaded.
4. Put 0200 in the switch register, press the LOAD ADD and START keys in that order. This causes the Keyboard Monitor, USR, and ABSLDR to be loaded on disk with a directory. The computer should then halt with 7777 in the accumulator lights. If the lights do not contain 7777, go back to step 2.
5. At this point the Binary Loader is still in core and is used to load the PS/8 Command Decoder tape. Place the tape in the reader, put 7777 in the switch register, press LOAD ADD, depress switch register bit 0, and press the START key. When the tape is loaded, the accumulator should be zero (all lights off). If any accumulator lights are on, a checksum error has occurred, and the tape should be reloaded.

6. Put 0200 in the switch register, press LOAD ADD and START keys in that order. This causes the Command Decoder and ODT to be written onto the disk and the Keyboard Monitor is brought into core. The system does a carriage return/ line feed and prints a dot at the left margin of the paper.

PS/8 is now up and running. Using ABSLDR, the user should then load the various system programs.

1. PIP Tape DEC-P8-PWXB-PB

   Place the PIP binary tape into the high-speed reader. Type the following (where . and * are printed by the Keyboard Monitor and Command Decoder, respectively)

```
.R ABSLDR              (RETURN key)
*PTR:=13ØØØ(89)$       ($=ALT MODE key)
```

   When ↑ is printed, type any character on the keyboard, and the tape is read. The Keyboard Monitor responds with a dot. Type:

```
.SAVE SYS PIP          (RETURN key)
```

   PIP has been saved on the system device.

2. EDITOR Tape DEC-P8-ESAB-PB

   Place the EDITOR binary tape in the reader. Load as follows:

```
.R ABSLDR              (RETURN key)
*PTR:(9)$              ($=ALT MODE key)
```

   When ↑ is printed, type any keyboard character to read the tape. The EDITOR is read and the Keyboard Monitor responds with a dot. Type:

```
.SAVE SYS EDIT         (RETURN key)
```

   The EDITOR is now written on the system device.

3. PAL8 Tape DEC-P8-ASAB-PB

   Place the PAL8 binary tape in the reader. Load as follows:

```
.R ABSLDR              (RETURN key)
*PTR:(9)$              ($=ALT MODE key)
```

   When ↑ is printed, type any keyboard character to load the tape. The tape is read and the Keyboard Monitor responds by printing a dot. Type:

```
.SAVE SYS PAL8         (RETURN key)
```

   PAL8 is now written onto the system device.

4. FORTRAN Tape DEC-P8-KFXB-PB

Place the FORTRAN binary tape into the reader. Load as follows:

```
.R ABSLDR          (type RETURN key)
*PTR:/S$           ($=ALT MODE key)
```

When ↑ is printed, type any keyboard character to initiate reading the tape. When loading is complete, Monitor responds with a dot. Type:

```
.SAVE SYS FORT     (RETURN key)
```

FORT has been saved on the system device.

5. SABR Tape DEC-P8-ARXB-PB

Place the SABR binary tape into the reader. Load as follows:

```
.R ABSLDR          (RETURN key)
*PTR:/S$           ($=ALT MODE key)
```

When ↑ is printed, type any keyboard character to initiate reading the tape. When loading is complete, Monitor responds with a dot. Type:

```
.SAVE SYS SABR     (RETURN key)
```

SABR has been saved on the system device.

6. Linking Loader Tape DEC-P8-LLXB-PB

Place the Linking Loader binary tape in the reader. Load as follows:

```
.R ABSLDR          (RETURN key)
*PTR:(9)$          ($=ALT MODE key)
```

When ↑ is printed, type any keyboard character to load the tape. Monitor responds with a dot. Type:

```
.SAVE SYS LOADER   (RETURN key)
```

The Linking Loader has been saved on the system device.

7. LIB8 Tapes DEC-P8-SFXB-PB, DEC-P8-SYXB-PB

Place the Library Setup (LIBSET) binary tape in the reader. Load as follows:

```
.R ABSLDR          (RETURN key)
*PTR:/G=126ØØ      (RETURN key)
```

When ↑ is printed, type any keyboard character to load the tape. When the first tape is loaded, the system responds by printing an asterisk. Now place the LIB8 Relocatable binary tape in the reader and type:

```
*/S$               ($=ALT MODE key)
```

The tape is read, and a LIB8.RL file is created on the system device.

8. CONVRT Tape DEC-P8-SUTB-PB

Place the CONVRT binary tape in the reader. Load as follows:

```
.R ABSLDR            (RETURN key)
*PTR:(9)$            ($=ALT MODE key)
```

When ↑ is printed, type any keyboard character to load the tape. After loading, Monitor responds with a dot. Type:

```
.SAVE SYS CONVRT     (RETURN key)
```

CONVRT is now written onto the system device.

That completes the building of the PS/8 system.

8.3 LOADING PS/8 ON A DISK SYSTEM FROM DECTAPE

Follow steps 1 to 5 in section 8.2 on loading PS/8 on a disk system from paper tape.

Put the DECtape containing the system library programs onto DECtape unit 1 (the unit number does not matter, but must be consistent with the commands given later). When the Keyboard Monitor prints a dot at the left margin, type the following commands:

```
.GET DTA1 PIP
.SAVE SYS PIP
.START
*EDIT.SV<DTA1:EDIT.SV  /I
*PAL8.SV<DTA1:PAL8.SV  /I
*FORT.SV<DTA1:FORT.SV  /I
*SABR.SV<DTA1:SABR.SV  /I
*LOADER.SV<DTA1:LOADER.SV  /I
*LIB8.RL<DTA1:LIB8.RL/I
*CONVRT.SV<DTA1:CONVRT.SV  /I
*↑C
.
```

When each of the system programs is loaded, the Command Decoder returns for further input and the user types CTRL/C to return to the Keyboard Monitor. All system library programs are now loaded and available for use.

8.4 DISK BOOTSTRAPS

Once a PS/8 System has been built on a disk, it may occasionally be necessary to start (or bootstrap) the system into operation when

nothing is in core memory. This procedure is also recommended when a program bug has been encountered such that the faulty program might have changed the contents of any location above 7577 in either field 0 or field 1.

The DF32 and RF08 bootstraps are as follows:

Location	Instruction
775Ø	76ØØ
7751	66Ø3
7752	6622
7753	5352
7754	5752

After depositing the bootstrap, set the switch register to 7750 press the LOAD ADD and START keys in that order. The PS/8 Keyboard Monitor should respond with a dot.

The RK8 bootstrap is as follows:

Location	Instruction
ØØ3Ø	6733
ØØ31	5Ø31

Set the switch register to 0030, press the LOAD ADD and START keys in that order.

8.5 RESTART LOCATIONS

If a PS/8 system should ever halt and cease any apparent response to the user, the system can be restarted by setting the switch register to either 76ØØ or 76Ø5 in field Ø, press LOAD ADD and then the START key.

Restarting at location 76ØØ saves the contents of core which are again available when the system resumes operation. Restarting at location 7605 does not preserve the contents of core and therefore saves time on a DECtape system.

8.6 PS/8 on the PDP-12

The PS/8 system provided for the PDP-12 consists of two LINCtapes. One LINCtape contains the source of PS/8 and CONFIG; the other contains the binary programs needed to use the system. This version of PS/8 will run on LINCtape formatted to either 128 or 129 words per block.

8.6.1 Bootstrapping the System

Either tape can be mounted on LINCtape unit Ø, as the system is present on both tapes. The tape unit should not be WRITE ENABLED. Set the console switches as follows:

left switches: 0700
right switches: 0000

Press the I/O PRESET switch in LINC mode. Depress the DO switch. When the processor halts, depress the START 20 switch.

This reads and starts the Keyboard Monitor. At this point the tape on LINCtape unit Ø should be WRITE ENABLED. From this point the operation of the system is exactly as in a standard PDP-8 system.

8.6.2 System Tape

The system LINCtape contains the source of PS/8 and CONFIG. The user can edit and assemble CONFIG to create special device handlers, etc. This version of CONFIG is fully commented.

8.6.3 Binary Tape

The binary LINCtape also contains the PS/8 system. In addition to the standard binaries of the system programs, various versions of CONFIG are already included:

a. LTA.BN
   This is the standard LINCtape system, including LINCtape handlers, where LINCtape unit Ø is the system device.
b. DF32.BN
   Contains LINCtape handlers, two DF32 disks are the system device.
c. RF08.BN
   Contains LINCtape, two RF08 disks are the system device.
d. RK8.BN
   Contains LINCtape, a single RK8 disk is the system device.

APPENDIX A

ASCII-1968 CHARACTER SET

Character	Code	Character	Code	Character	Code	
A	301	0	260	!	241	
B	302	1	261	"	242	
C	303	2	262	#	243	
D	304	3	263	$	244	
E	305	4	264	%	245	
F	306	5	265	&	246	
G	307	6	266	'	247	
H	310	7	267	(	250	
I	311	8	270	)	251	
J	312	9	271	*	252	
K	313			+	254	
L	314			,	254	
M	315			-	255	
N	316	<	274	.	256	
O	317	>	276	/	257	
P	320	/	337	:	272	
Q	321	_	337	;	273	
R	322	^	336	=	275	
S	323	↑	336	?	277	
T	324			[	333	(SHIFT/M)
U	325			\	334	(SHIFT/L)
V	326			]	335	(SHIFT/K)
W	327			BELL	207	
X	330			TAB	211	
Y	331			LINE FEED	212	
Z	332			CARRIAGE-RETURN	215	
				SPACE	240	
				RUBOUT	377	

Note 1: The separator character between output and input files (in lines to the Command Decoder) can be either < or ← . In preliminary versions it was ← .

Note 2: Due to a change in the ASCII character set, the characters ^ (circumflex) and _ (underline) replace ↑ (up-arrow) and ←(back-arrow), respectively.

PUNCHED CARD CODES

Zone	Punches	Character
		space
	1	1
	2	2
	3	3
	4	4
	5	5
	6	6
	7	7
	8	8
	9	9
	8-2	:
	8-3	#
	8-4	@
	8-5	'
	8-6	=
	8-7	"
0		Ø
0	1	/
0	2	S
0	3	T
0	4	U
0	5	V
0	6	W
0	7	X
0	8	Y
0	9	Z
0	8-2	]
0	8-3	,
0	8-4	%
0	8-5	←
0	8-6	>
0	8-7	?
11		-
11	1	J
11	2	K
11	3	L
11	4	M
11	5	N
11	6	O

Zone	Punches	Character
11	7	P
11	8	Q
11	9	R
11	8-2	!
11	8-3	$
11	8-4	*
11	8-5	)
11	8-6	;
11	8-7	\
12		&
12	1	A
12	2	B
12	3	C
12	4	D
12	5	E
12	6	F
12	7	G
12	8	H
12	9	I
12	8-2	[
12	8-3	.
12	8-4	<
12	8-5	(
12	8-6	+
12	8-7	↑

1. The card reader handler works with either a punched card reader or a mark-sense card reader.
2. The card reader handler works with any length card up to 80 characters.
3. NOTE: These are 029 card codes.

APPENDIX B

PS/8 ERROR MESSAGE SUMMARIES

Summary of Keyboard Monitor Error Messages

Message	Meaning
SYSTEM IO ERROR	An error occurred while doing I/O to the system device.
MONITOR ERROR 5 AT XXXXX	An error occurred while doing I/O to the system device. System device may not be WRITE enabled.
XXXX?	Where XXXX is not a legal command. For example, if the user typed "HELLO", the system responds "HELLO"?.
TOO FEW ARGS	An important argument has been omitted from a command.
name NOT AVAILABLE	Device name given in an ASSIGN, SAVE, RUN, or GET command was not available.
name NOT FOUND	The file name given was not found on the device indicated or the user tried to input from an output only device (see section 2.3.3 or 2.3.7).
BAD CORE IMAGE	The file requested was not a core image file (see section 2.3.3 or 2.3.7).
USER ERROR Ø AT xxxx	Input error detected while loading program. xxxx can be any location.
BAD ARGS	Arguments to SAVE command inconsistent (see section 2.3.4).
ILLEGAL ARG.	SAVE command not expressed correctly, illegal syntax (see section 2.3.4).
SAVE ERROR	An I/O error occurred while saving the program. Program is intact in core.
MONITOR ERROR 2 AT xxxx	Attempt made to output to a write locked device, usually DECtape.
NO!!	Attempt to restart a program which modifies itself (see 2.3.5).
BAD DATE	Improper syntax for date entry (see section 2.3.9).

Summary of Command Decoder Error Messages

Message	Meaning
ILLEGAL SYNTAX	The command line was formatted incorrectly.
TOO MANY FILES	More than three output files or nine input files were specified.
name DOES NOT EXIST	The device with the name given could not be found in the system tables.
name NOT FOUND	The file with the name given does not exist on the device indicated.

Summary of Symbolic Editor Error Messages

Minor errors are indicated by a question mark (?) printed at the left margin of the teleprinter paper. A ? is caused by an Editor command string error, attempt to execute a text type command without assigning a device, or a search for an unfound string.

Major errors cause the system to leave the Editor and return to the Keyboard Monitor. These messages are of the form:

```
? n ↑C
```

where n is an error code and ↑C indicates a return to the Keyboard Monitor. The meanings of the error codes are listed below:

Error Code	Meaning
Ø	Editor failed in reading a device.
1	Editor failed in writing onto a device.
2	File close error occurred.
3	File open error occurred.
4	Device handler error occurred.

When the output device is full and a write operation is attempted on that device, the output file is automatically closed and the message:

```
FULL
*
```

is printed. Control returns to the Command Decoder to obtain I/O specifications for the remainder of the output file. See page 4-14 for additional details.

Summary of PAL-8 Error Messages

Error messages for PAL-8 are the same as for 4K PAL-D and 8K PAL-D. The format of the message is:

ERROR CODE ADDRESS

where ERROR CODE is a two letter error designation code and ADDRESS is either the absolute octal address where the error occurred or the address of the error relative to hee last symbolic tag on the current page.

Error Code	Meaning
BE	Internal tables have overlapped, decrease the level of literal nesting or number of current page literals used prior to this point on the page.
DE	Device error detected during I/O operation. Control returns to the Keyboard Monitor.
IC	Illegal character, character is ignored, assembly continues.
ID	Illegal redefinition of a symbol. Symbol is not redefined.
IE	Illegal equals, may indicate an undefined symbol at that point.
II	Illegal indirect reference off page.
LD	Absolute Loader cannot be found on system device.
PE	Current non-zero page exceeded. See page 5-8.
PH	Phase error, missing $ or improper < or > .
SE	Symbol table exceeded, assembly terminated and control returns to the Keyboard Monitor.
US	Undefined symbol.
ZE	Page zero exceeded.

Summary of PIP Error Messages

Message	Meaning
NO ROOM FOR OUTPUT FILE	Self explanatory; either room on device or room in directory is lacking.
LINE TOO LONG IN FILE #n	In ASCII mode a line has been found greater than 140 characters. Be sure the transfer is really in ASCII mode. n is the number of the file in the input file list.
OUTPUT ERROR	Output error, possibly a write locked device, parity error, or attempt to output to the paper tape reader.
ERROR DELETING FILE	The user tried to delete a file that does not exist.
INPUT ERROR, FILE #n	An input error occurred while reading file n in the input file list.
CAN'T OPEN OUTPUT FILE	Output file on read-only device or else there is no name specified for the file.
DEVICE #n NOT A DIRECTORY DEVICE	Error message given by directory listing options when an incorrect device designation is made. n is the number of the device in the input list.
PREMATURE END OF FILE, FILE #n	/B option, incomplete binary input of file n in the input file list (ran out of input before finding trailer tape, for example).
ILLEGAL BINARY INPUT, FILE #n	Self explanatory; n is the number of the file in the input file list.
BAD DIRECTORY ON DEVICE #n	Directory listing error, the system is trying to read the directory of a blank device. n is the number of the device in the input list.
DIRECTORY ERROR	/S error; an error has occurred while reading or writing the directory during an /S option. The option is aborted; output is likely to be garbled.
IO ERROR - CONTINUING	/S error, error in copying a file. The /S option continues.
ARE YOU SURE?	/S message, respond with a Y if you are sure you want to compress the files.

Summary of PIP Error Messages (Cont.d)

Message	Meaning
SORRY - NO INTERRUPTIONS	/S message given if C is typed while compressing a device onto itself. The /S option continues.
NO ROOM - CONTINUING	/S message given when the output device cannot contain all files on the input device. The message is printed once for each file which will not fit onto the output file.

Summary of ABSLDR Error Messages

Message	Meaning
I/O ERROR FILE #n	An I/O error has occurred in input file number n.
BAD INPUT, FILE n	Attempt to load non-binary file as file number n.
BAD CHECKSUM, FILE #n	File number n had a checksum error.
NO INPUT	No input file was found on the designated device.
NO /I!	Use of /I is prohibited at this point.

Summary of CONVRT Error Messages

FILE OPEN ERR	An output file could not be opened on the specified device.
OUT DEV FULL	There is no more room on the output device. Something must be deleted on the output device before CONVRT can work properly.
INPUT DEV WRONG	The input device specified is not a DECtape, or SYS was specified as input.
INPUT READ ERR	The DECtape read routine detected some error while reading the input tape.
IN FILE NOT FOUND	The input file was not found, or none was specified.
BAD EOF	A zero link was detected before the logical End of File. The output file is closed at this point.
FILE CLOSE FAILED	An error occurred in closing the output file.
OUTPUT WRITE ERR	Device handler detected an error in transferring data.
OUT DEV HANDLER ERR	Output is inhibited, usually because no output device has been specified.

Summary of FORTRAN Error Messages

FORTRAN Compiler Error Messages

ARITHMETIC EXPRESSION TOO COMPLEX
EXCESSIVE SUBSCRIPTS
ILLEGAL ARITHMETIC EXPRESSION
ILLEGAL CONSTANT
ILLEGAL CONTINUATION
ILLEGAL EQUIVALENCING
ILLEGAL OR EXCESSIVE DO NESTING
ILLEGAL STATEMENT
ILLEGAL STATEMENT NUMBER
ILLEGAL VARIABLE
MOXED MODE EXPRESSION
SYMBOL TABLE EXCEEDED
SYNTAX ERROR (usually indicated illegal punctuation)

If an error is discovered in the user's FORTRAN program, the Compiler prints the incorrect line, followed by an error message. Although Compiler output will be suppressed, the rest of the user's program is read, and additional error messages are printed where necessary.

The following error messages have been added to the PS/8 version of FORTRAN:

Message	Explanation
I/O	A device handler has signalled an I/O error.
NO ROOM FOR OUTPUT	The file FORTRN.TM cannot fit on the system device.
SABR.SV NOT FOUND	The SABR Assembler is not present on the system device.
NO END STATEMENT	The input to the Compiler has been exhausted.
COMPILER MALFUNCTION	The meaning of this message has been extended to cover various unlikely monitor errors.

During execution, the various library programs check for certain errors and print error messages in the form:

XXXX ERROR AT LOC NNNNN

where XXXX is the error code and NNNNN is the location of the error.

Error Code	Meaning
	The following errors are fatal and cause a return to the Keyboard Monitor.
ALOG	Attempt to compute log of negative number.
IOER	One of the following has occurred: 1) Device independent input or output attempted without /I or /O options, 2) Bad arguments to IOPEN or OOPEN, or 3) Transmission error while doing I/O.
CHER	File specified as argument to CHAIN not found on system device.
FMT1	Invalid Format Statement
	The following input errors are fatal unless input is coming from the Teletype, in which case the entire READ statement is tried again.
FMT2	Illegal character in I format.
FMT3	Illegal character in F or E format
	The following errors do not terminate execution of the user's program.
DIVZ	Division by zero - very large number is returned.
EXP	Argument to EXP too large - very large number is returned.
OVFL	Floating point overflow - very large number is returned.
FLPW	Negative number raised to floating point power - absolute value taken.
SQRT	Attempt to take square root of negative number - absolute value used.
FIX	Attempt to fix a number >2047; 2047 is returned.

To pinpoint the location of a library program execution error:

a. Determine, from the storage map, the next lowest numbered location (external symbol) which is the entry point of the program or subprogram containing the error.

b. Subtract, in octal, the entry point location of the program or subprogram containing the error from the location of the error indicated in the error message.

c. From the assembly symbol table, determine the relative address of the external symbol found in step a and add that relative address to the result of step b.

d. The sum of step c is the relative address of the error, which can then be compared with the relative addresses of the numbered statements in the program.

Summary of 8K SABR Error Messages

During assembly, error messages are printed at the Teletype as they occur, in the form:

```
C  AT  LOC  +0004
```

which means that an error of type C has occurred at the fourth instruction after the location tag LOC. The line count includes comment lines and blank lines. The following messages can occur:

Error Code	Meaning
A	Too many or too few ARG statements follow a CALL statement.
C	An illegal character appears on the line.
D	A device handler has returned a fatal error condition.
L	/L or /G option was indicated, but the LOADER.SV file does not exist on the system device.
M	A symbol is multiply defined. Listings of programs with multiple definitions have unmarked errors.
I	An illegal syntax has been used, one of the following: a pseudo-op with improper arguments a quote mark with no argument a non-terminated text-string an improper address an illegal combination of micro-instructions
E	There is no END statement.
S	Either the symbol table has overflowed, common storage has been exhausted, more than 64 different user-defined symbols occurred in a core page, or more than 64 external symbols have been declared. Could also indicate a system error such as overflowed output file.
U	No symbol table is being produced, but there is at least one undefined symbol in the program.
UNDF	Undefined symbol, printed in the symbol table listing.

Summary of Linking Loader Error Messages

The Linking Loader gives error messages in the form:

ERROR nnnn

The different values of the nnnn error code are listed below:

Error Code	Meaning
0000	/I or /O specified too late.
0001	Symbol table overflow, more than 64 subprogram names.
0002	Program will not fit into core.
0003	Program with largest common storage area was not loaded first.
0004	Checksum error in input tape.
0005	Illegal relocation code.
0006	An output error has occurred while reading a binary file.
0010	No starting address has been specified and there is no entry point named MAIN.
0011	Input device handler requires two pages and will not fit into core where device independent I/O is being done.
0012	I/O error on system device.

APPENDIX C

PERMANENT SYMBOL TABLE FOR PAL-8 AND 8K SABR

The following are the most commonly used elements of the PDP-8 instruction set. For that reason they are found in the permanent symbol table within the assemblers. These instructions are already defined within the computer. For additional information on these instructions and for a description of the symbols used when programming other, optional, I/O devices, see the 1970 Small Computer Handbook, available from the DEC Program Library.

INSTRUCTION CODES

Mnemonic	Code	Operation
Memory Reference Instructions		
AND	0000	Logical AND
TAD	1000	Two's complement add
ISZ	2000	Increment and skip if zero
INC[1]	2000	Nonskip ISZ
DCA	3000	Deposit and clear AC
JMS	4000	Jump to subroutine
JMP	5000	Jump
Group 1 Operate Microinstructions		
OPR[2]	7000	Same as NOP
NOP	7000	No operation
IAC	7001	Increment AC
RAL	7004	Rotate AC and link left one
RTL	7006	Rotate AC and link left two
RAR	7010	Rotate AC and link right one
RTR	7012	Rotate AC and link right two
CML	7020	Complement link
CMA	7040	Complement AC
CLL	7100	Clear link
CLA	7200	Clear AC

[1]Not present in PAL-8.

[2]Not present in 8K SABR.

INSTRUCTION CODES (Cont'd)

Mnemonic	Code	Operation
Group 2 Operate Microinstructions		
HLT	7402	Halts the computer
OSR	7404	Inclusive OR SR with AC
SKP	7410	Skip unconditionally
SNL	7420	Skip on nonzero link
SZL	7430	Skip on zero link
SZA	7440	Skip on zero AC
SNA	7450	Skip on nonzero AC
SMA	7500	Skip on minus AC
SPA	7510	Skip on positive AC (zero is positive)
Combined Operate Microinstructions		
CIA	7041	Complement and increment AC
STL	7120	Set link to 1
GLK[3]	7204	Get link (put link in AC, bit 11)
STA	7240	Set AC to -1
LAS[3]	7604	Load AC with SR
Program Interrupt		
IOT[3]	6000	
ION	6001	Turn interrupt processor on
IOF	6002	Turn interrupt processor off
Keyboard/Reader		
KSF	6031	Skip on keyboard flag
KCC[3]	6032	Clear keyboard flag and AC
KRS[3]	6034	Read keyboard buffer (static)
KRB	6036	Read keyboard buffer (dynamic)

[3]Not present in 8K SABR

Mnemonic	Code	Operation
Teleprinter/Punch		
TSF	6041	Skip on teleprinter flag
TCF[4]	6042	Clear teleprinter flag
TPC[4]	6044	Load teleprinter and print
TLS	6046	Load teleprinter sequence
High Speed Reader		
RSF	6011	Skip on reader flag
RRB	6012	Read reader buffer and clear reader flag
RFC	6014	Reader fetch character
High Speed Punch		
PSF	6021	Skip on punch flag
PCF[4]	6022	Clear on punch flag
PPC[4]	6024	Load punch buffer and punch character
PLS	6026	Load punch buffer sequence
DECtape Transport Type TU55 and DECtape Control Type TC01[4]		
DTRA	6761	Contents of status register is ORed into AC bits 0-9
DTCA	6762	Clear status register A, all flags undisturbed
DTXA	6764	Status register A loaded by exclusive OR from AC. If AC bit 10=0, clear error flags; if AC bit 11=0, DECtape control flag is cleared
DTSF	6771	Skip if error flag is 1 or if DECtape control flag is 1
DTRB	6772	Contents of status register B is ORed into AC
DTLB	6774	Memory field portion of status register B loaded from AC bits 6-8.

[4]Not present in 8K SABR.

Mnemonic	Code	Operation
Disk File and Control, Type DF 32[5]		
DCMA	6601	Clear disk memory request and interrupt flags
DMAR	6603	Load disk from AC, clear AC read into core, clear interrupt flag.
DMAW	6605	Load disk from AC, write onto disk from core, clear interrupt flag.
DCEA	6611	Clear disk extended address and memory address extension register
DSAC	6612	Skip if address confirmed flag = 1
DEAL	6615	Clear disk extended address and memory address extension register and load same from AC
DEAC	6616	Clear AC, load AC from disk extended address register, skip if address confirmed flag = 1
DFSE	6621	Skip if parity error, data reques late, or write lock switch flag = 0 (no error)
DFSC	6622	Skip if completion flag = 1 (data transfer completed)
DMAC	6626	Clear AC, load AC from disk memory address register
Memory Extension Control, Type 183[5]		
CDF N0	6201	Change to data field N
CIF N0	6202	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

[5]Not present in 8K SABR

PSEUDO-OPERATORS

The following is a list of the PAL-8 and 8K SABR pseudo-ops. The first section consists of those pseudo=ops which have counterparts in the other assembler. Below the blank space are the various pseudo-ops individual to the particular assembler.

PAL-8	8K SABR
DECIMAL	DECIMAL
OCTAL	OCTAL
PAUSE	PAUSE
I	I
$	$
PAGE	PAGE
EXPUNGE	ABSYM
FIXTAB	ARG
Z	BLOCK
FIELD	CALL
XLIST	COMMN
IFDEF	CPAGE
IFNDEF	DUMMY
IFZERO	EAP
IFNZRO	END
ENPUNCH	ENTRY
NOPUNCH	FORTR
ZBLOCK	IF
EJECT	LAP
TEXT	OPDEF
FIXMRI	REORG
	RETRN
	SKPDEF

NOTE: The symbols ACH, ACM, and ACL are also present in the permanent symbol table for 8K SABR. For details, see Chapter 14, Programming Languages.

New Instructions for the Basic PDP-8/E

The following instructions are present in every PDP-8/E, but not in the other members of the PDP-8 family. None of these instructions (other than NOP or CLA) is in either SABR or PAL-8.

Mnemonic	Code	Operation
SKON	6000	Skip if interrupt on and turn interrupt off
SRQ	6003	Skip if interrupt request
GTF	6004	Get interrupt flags
RTF	6005	Restore interrupt flags
SGT	6006	Skip if greater than flag
CAF	6007	Clear all flags
BSW	7002	Swap bytes in AC
NOP	7401	No operation
CLA	7601	Clear AC
MQL	7421	Load MQ from AC, then clear AC
MQA	7501	Inclusive OR, MQ with AC
DAD	7563	Double precision add
DST	7565	Double precision store
DCM	7567	Double precision two's complement
NMU	7451	Absolute normalize
SAM	7453	Subtract AC from MQ
CAM	7621	Swap AC and MQ
ACL	7701	Load MQ into AC (conflicts with SABR permanent symbol, change one or the other)
CLA SWP	7721	Load AC from MQ, clear MQ

APPENDIX D

PS/8 DEMONSTRATION PROGRAM

The following pages contain a demonstration of the PS/8 system taken from a teleprinter on-line with PS/8. The teleprinter paper is coded with letters and brackets (to the left of the teleprinted material). The letters correspond to the textual explanations on the facing page.

The demonstration exhibits the use of some utility programs, the Editor, and FORTRAN Compiler, and will give the reader a feeling for PS/8 interaction.

```
A  •R PIP
   *DTA1:</Z=1
B  */E↑C

   •DATE 10/19/70

C-  •ASSIGN DTA1 IN
D  •R PIP
   *IN/E
   IN NOT FOUND
   *IN:/E

E   10/19/70
   <EMPTY> 730
    730 FREE BLOCKS

F-  *↑C

G  •GET SYS EDIT

   •SAVE IN EDIT 0-5000;200=2001

   •R FORT
   *IN:FORT2<FORT2/G
   /10      B(I)=FLOAT(I)*A(I
            ↑
   ILLEGAL ARITHMETIC EXPRESSION
H  /603     FORMAT('TO REPEAT, TYPE A CARRIAGE RETURN'//)
            ↑
   NO END STATEMENT

   •RUN IN EDIT
I  *IN:FORT3<FORT2

   #Y

   #/:0043
   #$10     'L
   10       B(I)=FLOAT(I)*A(I
J
   #•S
   10       B(I)=FLOAT(I)*A(I)

   #•L
   10       B(I)=FLOAT(I)*A(I)

   #/L
   603      FORMAT('TO REPEAT, TYPE A CARRIAGE RETURN'//)
K
   #A
            END
```

A The user calls PIP into core from the Keyboard Monitor. The Command Decoder prints a star (*) and awaits an input line. The first input line gives the command to zero the DECtape on unit 1, specifying one additional information word in the directory.

B By typing CTRL/C the user returns control to the Keyboard Monitor and uses the DATE command to set the system date to October 19, 1970.

C The ASSIGN command is used to give DTA1 the additional name IN. All subsequent references to IN refer to DECtape unit 1.

D PIP is called again to list the directory of DECtape unit 1. The user gets the error "IN NOT FOUND" because he forgot to put a colon after IN.

E This is the extended directory listing of DTA1.

F The Command Decoder returns to accept another PIP command. The user types CTRL/C to return to the Keyboard Monitor.

G The Keyboard Monitor GET and SAVE commands are used to copy EDIT from the system device to DECtape unit 1.

H The FORTRAN Compiler is run to compile and execute the program FORT2. However, the program has two errors in it. FORTRAN returns control to the Keyboard Monitor after compiling the program and printing the error messages.

I The program EDIT, located on DECtape unit 1, is used to correct the errors in the FORTRAN program. The old program FORT2 is input to the Editor and a new program, FORT3, is written by the Editor onto DECtape unit 1.

J The string search feature of the Editor is used to find line 10 and correct it.

K An END statement is added to the program.

```
#L

C       THIS IS A DEMO OF SOME OF THE NEW FEATURES
C       IN PS/8 FORTRAN.        SPECIFICALLY, IMPLIED DO LOOPS
C       AND DIRECT INSERTION OF SABR CODE.

C       THE PROGRAM PERFORMS A SIMPLE TRANSFORMATION ON
C       A TEN ELEMENT ONE DIMENSIONAL ARRAY (A(I)), PUTTING
C       THE RESULTING ARRAY IN B.       BOTH THE OLD AND NEW
C       ARRAYS ARE PRINTED, AND THE USER HAS THE OPTION OF
C       REPEATING THE PROGRAM OR TERMINATING EXECUTION.

        DIMENSION A(10),B(10)
05      WRITE(1,600)
        READ(1,500) (A(I),I=1,10)
        WRITE(1,601)
        WRITE(1,500) (A(I),I=1,10)
        DO 10 I=1,10
10      B(I)=FLOAT(I)*A(I)
        WRITE(1,602)
        WRITE(1,500) (B(I),I=1,10)
        WRITE(1,604)

C       THE SABR CODE FOLLOWING LOOKS FOR A CARRIAGE
C       RETURN CHARACTER TO INITIATE REPEATING THE
C       PROGRAM.        ANY OTHER CHARACTER TERMINATES THE PROGRAM.

SX,     KSF
S       JMP X
S       KRB
S       TAD MYES
S       SZA
S       JMP \20
        GO TO 05
SMYES,  -215

20      CALL EXIT

500     FORMAT(F6.2)
600     FORMAT('ENTER 10 NUMBERS IN F6.2 FORMAT'/)
601     FORMAT('THE ORIGINAL ARRAY IS:'/)
602     FORMAT('THE TRANSFORMED ARRAY IS:'/)
603     FORMAT('TO REPEAT, TYPE A CARRIAGE RETURN'//)
        END

#$(1,604)'S
        WRITE(1,604\3)

#.L
        WRITE(1,603)

#E
```

L The user makes the Editor list the entire FORTRAN program.

M Note the use of implied DO loops in the READ and WRITE statements. This is a new feature of PS/8 FORTRAN.

N An S in column 1 of a FORTRAN line indicates that the line contains SABR code. This is an important new feature of PS/8 FORTRAN.

O At the end of the program CALL EXIT is used to return control to the Keyboard Monitor.

P The string search feature of the Editor is used to correct another error discovered when the program was listed.

Q The E command to the Editor closes the file and returns to the Keyboard Monitor.

```
R-   .AS DTA1 OUT
S {  .R FORT
     OUT:FORT3<OUT:FORT3/G

     ENTER 10 NUMBERS IN F6.2 FORMAT

     1.0
     2.0
     3.0
     4.0
     5.0
     6.0
     7.0
     8.0
     9.0
     10.0
     THE ORGINAL ARRAY IS:

       1.00
       2.00
       3.00
       4.00
T      5.00
       6.00
       7.00
       8.00
       9.00
      10.00
     THE TRANSFORMED ARRAY IS:

       1.00
       4.00
       9.00
      16.00
      25.00
      36.00
      49.00
      64.00
      81.00
     100.00
U-   TO REPEAT, TYPE A CARRIAGE RETURN

V {  .DEAS
     .AS DTA1 X
     .R PIP
     *X:/L

      10/19/70
W    EDIT   .SV      10 10/19/70
     FORT3            3 10/19/70
     FORT3  .RL       3 10/19/70
      714 FREE BLOCKS
```

R The ASSIGN command is used to change the assigned name of DTA1 from IN to OUT.

S The FORTRAN Compiler is called again. The /G option is used to cause automatic loading and execution of the FORTRAN program. In addition, an output relocatable binary file named FORT3 is saved by SABR on DECtape unit 1.

T The results of the FORTRAN program.

U After running the program once the user types something other than the RETURN key and exits to the Keyboard Monitor.

V The DEASSIGN command is used to delete all user assigned device names. The ASSIGN command is then used to give the name X to DTA1.

W PIP is run to give the directory listing of DECtape unit 1.

```
*TTY:<SYS:/L

 10/19/70
ABSLDR.SV          5
PIP    .SV        10
EDIT   .SV        10
FORT   .SV        25
SABR   .SV        24
LOADER.SV         10
LIB8   .RL        29
FORT2              3
 533 FREE BLOCKS
```

X

```
*FORT2</D
*FORT3<X:FORT3
*^C

.
```

Y

X Next, PIP is used to print the directory of the system device. Note that this directory has no dates. This is because no additional information words were specified in the directory of the system device.

Y PIP is then used to delete the old file FORT2, and then to copy the ASCII file FORT3 from DECtape unit 1 to the device DSK (which is the same as SYS in this example). Finally, CTRL/C is typed to return control to the Keyboard Monitor before leaving.

INDEX

HOW TO OBTAIN SOFTWARE INFORMATION

Announcements of new and revised software, as well as programming notes, software problems, and documentation corrections are published by Software Information Service in the following newsletters:

Digital Software News for the PDP-8 and PDP-12

Digital Software News for the PDP-9/15 Family

Digital Software News for the PDP-11

These newsletters contain information to update the cumulative

Software Performance Summary for the PDP-8 and PDP-12

Software Performance Summary for the PDP-9/15 Family

Software Performance Summary for the PDP-11

The appropriate edition of the Software Performance Summary is included in each basic software kit for new customers. Additional copies may be requested without charge.

Any questions or problems on the articles contained in these publications or concerning the use of Digital's software should be reported to the Software Specialist or Sales Engineer at the nearest Digital office.

New and revised software and manuals, and current issues of the Software Performance Summary are available from the Program Library. To place an order, write to

Program Library
Digital Equipment Corporation
146 Main Street, Building 1-2
Maynard, Massachusetts 01754

When ordering, include the code number and a brief description of the program or manual requested.

Digital Equipment Computer Users Society (DECUS) maintains a user library and publishes a catalog of available programs as well as the DECUSCOPE magazine for its members and non-members who request it. For further information, please write to:

DECUS
Digital Equipment Corporation
146 Main Street
Maynard, Massachusetts 01754

READER'S COMMENTS

Digital Equipment Corporation maintains a continuous effort to improve the quality and usefulness of its publications. To do this effectively we need user feedback – your critical evaluation of this manual.

Please comment on this manual's completeness, accuracy, organization, usability, and readability.

Did you find errors in this manual?

How can this manual be improved?

Other comments?

Please describe your position.

Name _____ Organization _____

Street _____ Department _____

City _____ State _____ Zip or Country _____

..... 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

Digital Equipment Corporation
Software Information Services
146 Main Street, Bldg. 3-5
Maynard, Massachusetts 01754