edusystem

EDUsystem - 50

USERS GUIDE

DEC-08-E50UA-A-D

digital

digital equipment corporation

EDUsystem - 50

USERS GUIDE

DEC-08-E50UA-A-D

August 1975

DIGITAL EQUIPMENT CORPORATION MAYNARD, MASSACHUSETTS 01754

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

The software described in this document is furnished under a license and may only be used or copied in accordance to the terms of such license.

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

The HOW TO OBTAIN SOFTWARE INFORMATION pages, located at the back of this document, explain the various services available to Digital software users.

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

The following are trademarks of Digital Equipment Corporation:

DIGITAL	DECsystem-10	MASSBUS
DEC	DECtape	OMNIBUS
PDP	DIBOL	OS/8
DECUS	EDUSYSTEM	PHA
UNIBUS	FLIP CHIP	RSTS
COMPUTER LABS	FOCAL	RSX
COMTEX	INDAC	TYPESET-8
DDT	LAB-8	TYPESET-11
DECCOMM		

LIMITED RIGHTS LEGEND

Contract No. _____

Contractor or Subcontractor: Digital Equipment Corporation

All the material contained herein is considered limited rights data under such contract.

CONTENTS

PREFACE			
CHAPTER 1		INTRODUCTION	1-1
	1.1	USER PROGRAMS	1-2
	1.2	USER FILES	1-2
	1.3	TSS/8 HARDWARE CONFIGURATION	1-3
CHAPTER 2		EDUSYSTEM 50 MONITOR	2-1
	2.1	CALLING THE MONITOR	2-6
	2.2	LOGGING INTO EDSYSTEM 50	2-8
	2.3	LOGGING OUT OF EDUSYSTEM 50	2-10
	2.4	SYSTEM LIBRARY PROGRAM CONTROL	2-13
	2.5	TYPING MONITOR COMMANDS	2-13
	2.6	CONTROLLING OUTPUT	2-13
	2.7	COMMUNICATING WITH OTHER USERS	2-14
	2.8	HUNG OUTPUT DEVICES	2-15
	2.9	SYSTEM STATUS REPORTS	2-15
	2.10	RESOURCE SHARING	2-16
	2.11	ERROR MESSAGES	2-20
CHAPTER 3		SYSTEM LIBRARY PROGRAMS	
	3.1	GENERAL FILE CHARACTERISTICS	3-2
	3.2	CONTROLLING THE EXECUTION OF SYSTEM LIBRARY PROGRAMS	3-5
	3.3	RETURNING TO THE MONITOR	3-6

CHAPTER 4 CALLING AND USING BASIC

4.1 BASIC 4-1

4.2 LANGUAGE FEATURES 4-2

4.2.1 Truncation Function, FIX (X) 4-2

4.2.2 ON GOTO Statement 4-2

4.2.3 SLEEP Statement 4-3

4.2.4 Comments 4-4

4.2.5 Blank Lines 4-4

4.2.6 Multiple Statements per Line 4-5

4.2.7 Editing BASIC Statements 4-5

4.2.8 Saving Compliled Programs 4-6

4.2.9 File Protection 4-7

4.2.10 Project-Programmer Numbers 4-7

4.2.11 Restricted Accounts 4-8

4.2.12 Catalog Format 4-8

4.2.13 Strings in BASIC 4-8

4.2.13.1 Reading String Data 4-9

4.2.13.2 Printing Strings 4-11

4.2.13.3 Inputting Strings 4-11

4.2.13.4 Line Input 4-12

4.2.13.5 Working with Strings 4-13

4.2.13.6 The CHANGE Statement 4-14

4.2.13.7 The CHR$ Function 4-16

4.2.14 Program Chaining 4-17

4.3 DISK DATA FILES 4-19

4.3.1 File Records 4-19

4.3.2 Opening a Disk File 4-21

	4.3.3	Reading/Writing Disk Files	4-21
	4.3.4	Closing/Deleting Disk Files	4-24
	4.4	DECTAPE DATA FILES	4-24
	4.4.1	DECtape File Records	4-25
	4.4.2	Opening a DECtape File	4-26
	4.4.3	Reading/Writing DECtape Files	4-27
	4.4.4	Closing DECtape Files	4-29
	4.4.5	Using DECtape Data Files with OS/8 FORTRAN	4-29
	4.5	LINE PRINTER OUTPUT	4-29
	4.6	INTERNAL DATA CODES	4-30
	4.6.1	Numeric Data	4-30
	4.6.2	String Data	4-31
	4.7	ERROR MESSAGES	4-32
CHAPTER 5		FOCAL	
	5.1	USING FOCAL COMMANDS	5-1
	5.2	FOCAL OVER VIEW	5-2
	5.3	NUMBERS	5-3
	5.4	VARIABLE NAMES	5-4
	5.5	ARITHMETIC OPERATIONS	5-5
	5.5.1	Priority of Arithmetic Operations	5-5
	5.5.2	Enclosures	5-6
	5.6	INPUT/OUTPUT COMMANDS	5-7
	5.6.1	TYPE Command	5-7
	5.6.2	ASK Command	5-8
	5.6.2.1	Text Output with ASK	5-9

5.7	COMMPUTATIONAL COMMAND (SET)	5-9
5.8	CONTROL COMMAND	5-10
5.8.1	GO or GOTO Command	5-10
5.8.2	IF Command	5-11
5.8.2.1	IF with Less Than Three Line Numbers	5-11
5.8.2.2.	Arithmetic Comparison with IF Command	5-12
5.8.3	DO Command	5-13
5.8.3.1	Nested DO	5-13
5.8.4	RETURN Command	5-14
5.8.5	QUIT Command	5-14
5.8.6	FOR Command	5-14
5.8.6.1	FOR with a DO	5-15
5.8.6.2	Nested FOR and DO	5-15
5.8.6.3	Subscript Variables	5-16
5.8.7	COMMENT or CONTINUE Command	5-17
5.9	EDIT COMMAND	5-17
5.9.1	WRITE OR WRITE ALL Command	5-17
5.9.2	ERASE and ERASE ALL Command	5-18
5.9.3	MODIFY Command	5-19
5.10	LIBRARY COMMANDS	5-21
5.10.1	LIBRARY SAVE Command	5-21
5.10.2	LIBRARY CALL Command	5-22
5.10.3	Error Messages with Library Commands	5-22
5.11	ESTIMATING PROGRAM LENGTH	5-23
5-12	DEBUGGING	5-24

5.12.1 Using the Error Diagnostics 5-24

5.12.2 Using the Trace Feature 5-25

5.13 FOCAL FUNCTIONS 5-25

5.13.1 Sine Function (FSIN) 5-26

5.13.2 Cosine Function (FCOS) 5-26

5.13.3 Exponential Function (FEXP) 5-27

5.13.4 Logarithm Function (FLOG) 5-27

5.13.5 Arctangent Function (FATN) 5-28

5.13.6 Square Root Function (FSQT) 5-28

5.13.7 Absolute Value Function (FABS) 5-28

5.13.8 Sign Part Function (FSGN) 5-29

5.13.9 Integer Part Function (FITR) 5-29

5.13.10 Random Number Function (FRAN) 5-30

5.14 FOCAL OUTPUT OPERATIONS 5-30

5.15 CONTROL CHARACTERS 5-31

CHAPTER 6 FORTRAN 6-1

6.1 CALLING FORTRAN-D 6-1

6.2 USING FORTRAN-D 6-2

6.3 LINE FORMAT 6-4

6.3.1 Statement Numbers 6-5

6.3.2 Statement Continuation Character 6-5

6.4 FORTRAN STATEMENTS 6-6

6.4.1 Comment Statements 6-6

6.4.2 Character Set 6-7

6.4.3 Constants 6-7

5.13 FOCAL FUNCTIONS

5.13.1 Sine Function (FSIN)

5.13.2 Cosine Function (FCOS)

5.13.3 Exponential Function (FEXP)

5.13.4 Logarithm Function (FLOG)

5.13.5 Arctangent Function (FATN)

5.13.6 Square Root Function (FSQT)

5.13.7 Absolute Value Function (FABS)

5.13.8 Sign Part Function (FSGN)

5.13.9 Integer Part Function (FITR)

5.13.10 Random Number Function (FRAN)

5.14 FOCAL OUTPUT OPERATIONS

5.15 CONTROL CHARACTERS

CHAPTER 6 FORTRAN

6.1 CALLING FORTRAN-D

6.2 USING FORTRAN-D

6.3 LINE FORMAT

6.3.1 Statement Numbers

6.3.2 Statement Continuation Character

6.4 FORTRAN STATEMENTS

6.4.1 Comment Statements

6.4.2 Character Set

6.4.3 Constants

6.4.3.1 Integer Constants

6.4.3.2 Real Constants

6.4.3.3 Fixed and Floating-Point Representation

6.4.3.1	Integer Constants	6-7
6.4.3.2	Real Constants	6-8
6.4.3.3	Fixed and Floating-Point Representation	6-8
6.4.4	Variables	6-10
6.4.4.1	Integer Variables	6-11
6.4.4.2	Real Variables	6-11
6.4.4.3	Scalar Variables	6-11
6.4.4.4	Array Variables	6-11
6.4.5	DIMENSION Statement	6-12
6.5	FORTRAN ARITHMETIC	6-13
6.5.1	Arithmetic Operators	6-13
6.5.1.1	Use of Parentheses	6-15
6.5.2	Arithmetic Expressions	6-16
6.5.3	Arithmetic Statements	6-17
6.5.3.1	Multiple Replacement	6-18
6.5.4	Functions	6-20
6.6	PROGRAM CONTROL STATEMENTS	6-21
6.6.1	END Statement	6-21
6.6.2	STOP Statement	6-21
6.6.3	PAUSE Statement	6-22
6.6.4	GO TO Statement	6-22
6.6.5	Example of Integer Summation	6-23
6.6.6	IF Statement	6-23
6.6.7	DO Loops	6-25
6.6.7.1	CONTINUE Statement	6-28
6.6.8	Computed GO TO	6-28

6.7	FORTRAN INPUT/OUTPUT	6-29
6.7.1	Data Formats	6-30
6.7.1.1	ASCII Coded Data	6-30
6.7.1.2	Binary Coded Data	6-30
6.7.2	Input/Output Statements	6-30
6.7.2.1	ACCEPT and TYPE Statements	6-31
6.7.2.2	Read and Write Statements	6-32
6.7.3	Variable Specification in I/O Statements	6-33
6.7.4	FORMAT Statements	6-35
6.7.5	The A Format Specification	6-36
6.7.6	Input Format	6-37
6.7.6.1	Integer Values --the I Format	6-37
6.7.6.2	Real Values -- the E Format	6-38
6.7.7	Output Formats	6-38
6.7.7.1	E and I Formats	6-38
6.7.7.2	FORMATS Control Specifications	6-39
6.7.7.3	Hollerith Output	6-40
6.8	INPLEMENTATION NOTES	6-40
6.8.1	Double Subscripts	6-40
6.8.2	Substatement Feature	6-42
6.8.3	Error Checking	6-43
6.8.4	FORTRAN-D Source Program Restrictions	6-44
6.8.5	FORTRAN-D Compiler and Operating System Core Map	6-44

	6.9	FORTRAN-D ERROR DIAGNOSTICS	6-45
	6.9.1	Compiler Compilation Diagnostics	6-45
	6.9.2	Compiler Systems Diagnostics	6-49
	6.9.3	Operating System Diagnostics	6-50
CHAPTER 7		PAL-D ASSEMBLER	7-1
	7.1	INTRODUCTION	7-1
	7.2	EDUSYSTEM 50 PAL-D	7-2
	7.3	SYNTAX	7-2
	7.3.1	Legal Characters	7-3
	7.3.2	Illegal Characters	7-4
	7.3.3	Format Effectors	7-4
	7.4	NUMBERS	7-6
	7.4.1	Arithmetic and Logical Operators	7-6
	7.4.2	Evaluating Expressions	7-7
	7.5	STATEMENTS	7-7
	7.5.1	Labels	7-8
	7.5.2	Operators	7-8
	7.5.3	Operands	7-8
	7.5.4	Comments	7-9
	7.6	SYMBOLS	7-9
	7.6.1	Symbol Distinction	7-9
	7.6.1.1	Permanent Symbols	7-9
	7.6.1.2	User-Defined Symbols	7-10
	7.6.2	Symbolic Address	7-11
	7.6.3	Symbolic Operators	7-12
	7.6.4	Symbolic Operands	7-12

7.6.5 Symbol Table 7-12

7.6.6 Direct Assignment Statements 7-13

7.7 ADDRESS ASSIGNMENTS 7-14

7.7.1 Current Address Indicator 7-17

7.7.2 Indirect Address 7-18

7.7.3 Autoindexing 7-20

7.7.4 Literals 7-21

7.8 INSTRUCTIONS 7-23

7.8.1 Memory Reference Instructions 7-23

7.8.1.1 Paging 7-24

7.8.1.2 Off-Page Referencing 7-25

7.8.2 Augmented Instructions 7-25

7.8.2.1 Operate Microinstructions 7-25

7.8.2.2 Input-Output Transfer Microinstructions 7-27

7-9 PSEUDO-OPERATORS 7-31

7.9.1 Current Location Counter 7-31

7.9.2 Extended Memory 7-31

7.9.3 RADIX Control 7-32

7.9.4 Listing Control 7-32

7.9.5 TEXT Facility 7-32

7.9.6 End of Program 7-33

7.9.7 End of File 7-33

7.9.8 Altering the Symbol Table 7-34

7.9.9 Internal Representation 7-34

	7.10	PROGRAM PREPARATION AND ASSEMBLER OUTPUT	7-35
	7.10.1	Program File	7-35
	7.10.2	Assembly	7-37
	7.10.3	Pass 1	7-37
	7.10.4	Pass 2	7-38
	7.10.5	Pass 3	7-38
	7.11	OPERATING THE PALD ASSEMBLER	7-39
	7.12	ERROR DIAGNOSTICS	7-42
CHAPTER 8		UTILITY PROGRAMS	8-1
	8.1	SYMBOLIC EDITOR	8-1
	8.2	LOADER	8-2
	8.3	OCTAL DEBUGGING TECHNIQUE (ODTHI)	8-6
	8.4	CATALOG (CAT)	8-7
	8.5	SYSTEM STATUS (SYSTAT)	8-9
CHAPTER 9		PROGRAMS FOR HANDLING DATA	9-1
	9.1	PERIPHERAL UTILITY TRANSFER ROUTINES (PUTR)	9-1
	9.1.1	PUTR Commands	9-1
	9.1.1.1	ZERO Commands	9-10
	9.1.1.2	DELETE Commands	9-11
	9.1.1.3	DIRECTORY Commands	9-11
	9.1.1.4	LIST Command	9-12
	9.1.1.5	TYPE Command	9-12
	9.1.1.6	PUNCH Command	9-13
	9.1.1.7	TAPE Command	9-13

9.1.1.8 EXIT Command 9-13

9.1.1.9 Special Notes 9-13

9.1.1.10 Default Propagation 9-15

9.1.1.11 DECtape and RKØ5 Disks 9-16

9.1.2 TSS/8 File Extensions 9-17

9.1.3 PUTR Switches 9-17

9.2 PERIPHERAL INTERCHANGE PROGRAM (PIP) 9-18

9.2.1 PIP Conventions 9-18

9.2.2 Paper Tape to Disk Transfers 9-19

9.2.3 Disk to Paper Tape Transfers 9-19

9.2.4 High-Speed Reader/Punch Assignments 9-20

9.2.5 BIN Format File Transfers 9-21

9.2.6 Moving Disk Files 9-21

9.2.7 Deleting Disk Files 9-21

9.3 COPY PROGRAM 9-22

9.3.1 Using and Calling COPY 9-22

9.3.2 Loading Files from DECtape 9-23

9.3.3 Saving Disk Files on DECtape 9-23

9.3.4 Listing Directories 9-24

9.3.5 Deleting Files 9-25

9.3.5.1 Deleting All Existing Files on a Device 9-25

9.3.6 Example of COPY Usage 9-26

9.3.7 BASIC File Transfers 9-28

9.3.8 Save Format File Transfers 9-28

CHAPTER 10		ADVANCED MONITOR COMMANDS	10-1
	10.1	INTRODUCTION	10-1
	10.2	CONTROL OF USER PROGRAMS	10-3
	10.3	DEFINING DISK FILES	10-4
	10.3.1	Creating a Disk File	10-5
	10.3.2	Opening and Closing a File	10-5
	10.3.3	Extending, Reducing, and Renaming a Disk File	10-6
	10.3.4	Protection Codes	10-7
	10.3.5	Error Conditions	10-10
	10.4	SAVING AND RESTORING USER PROGRAMS	10-10
	10.5	UTILITY COMMANDS	10-14
CHAPTER 11		WRITING ASSEMBLY LANGUAGE PROGRAMS	11-1
	11.1	INTRODUCTION	11-1
	11.2	CONSOLE I/O	11-2
	11.3	FILES AND DISK I/O	11-6
	11.4	ASSIGNABLE DEVICES	11-14
	11.5	PROGRAM CONTROL	11-21
	11.6	PROGRAM AND SYSTEM STATUS	11-23
	11.7	PDP-8 COMPATIBILITY	11-28

APPENDIX A MONITOR COMMAND SUMMARY A-1

A.1 MONITOR COMMANDS A-1

A.1.1 Logging In and Out A-1

A.1.2 Device Allocation A-2

A.1.3 File Handling A-2

A.1.4 Control of User Programs A-3

A.1.5 Utility Commands A-4

APPENDIX B CHARACTER CODES B-1

APPENDIX C STORAGE ALLOCATION C-1

C.1 STORAGE MAP C-1

C.2 FILE DIRECTORIES C-1

APPENDIX D* ERROR MESSAGES D-1

D.1 BASIC ERROR MESSAGES

D.2 CAT ERROR MESSAGES

D.3 COPY ERROR MESSAGES

D.4 DECODE ERROR MESSAGES

D.5 FOCAL ERROR MESSAGES

D.6 FORTRAN-D COMPILER COMPILATION DIGNOSTICS

D.7 FORTRAN-D COMPILER SYSTEMS DIAGNOSTICS

D.8 FORTRAN-D OPERATING SYSTEM DIAGNOSTICS

D.9 LOADER ERROR MESSAGES

*NOTE: Information on the above can be found from page D-1 through D-27.

D.10 LOGID ERROR MESSAGES

D.11 MONITOR ERROR MESSAGES

D.12 NON-FATAL EXECUTION ERROR MESSAGES

D.13 PAL-D ERROR DIAGNOSTICS

D.14 PERIPHERAL INTERCHANGE PROGRAM (PIP) ERROR MESSAGES

D.15 PERIPHERAL UTILITY TRANSFER ROUTINES (PUTR) ERROR MESSAGES

NOTE: Information on the above can be found from page D-1 through D-27.

CHAPTER 1

INTRODUCTION

EduSystem 50 is a general purpose, time-sharing system for PDP-8 computers that offers up to 20 users a comprehensive library of System Programs. These programs provide facilities for editing, assembling, compiling, debugging, loading, saving, calling, and executing user programs on-line. An extended BASIC language provides users with the ability to use strings, files, and program chaining. Two higher-level languages, FOCAL and FORTRAN, are also provided. All languages and utilities may be used simultaneously. One group of users may be working in BASIC while another is using assembly language. EduSystem 50 serves all levels of users simultaneously.

By separating the central processing operations from time-consuming interactions with human users, the computer can, in effect, work on a number of programs simultaneously. Cycling between programs and giving only a fraction of a second at a time to each program or task, the computer can deal with many users seemingly at once. The appearance is created that each user has the computer to himself. The execution of various programs is done without their interfering with each other and without lengthy delays in the response to individual users.

The heart of EduSystem 50 is a complex of subprograms called the Monitor. The Monitor coordinates the operations of the various programs and user consoles, ensuring that the user is always in contact with his program. The EduSystem 50 Monitor allocates the time and services of the computer to the various users; it grants a slice of processing time to each job, and schedules jobs in sequential order to make most efficient use of the system disk. The Monitor handles user requests for hardware operations (reader, punch, etc), swaps (moves) programs between memory and disk, and manages the user's private files.

1.1 USER PROGRAMS

When the user is working with EduSystem 50, it appears to him as though he had his own 4K (4096 word) PDP-8 computer. He then has the capability of doing anything which can be done in a 4K computer plus the capabilities of the Monitor. Several users can run different programs at virtually the same time because the Monitor controls the scheduling of execution times. The Monitor brings a program into core from the disk, allows it to execute for a short time, and takes note of the state at which execution is stopped. The user is allotted a 4K block of core that contains his particular program; this 4K block is swapped (moved) from core onto a 4K area of disk when the Monitor needs to bring another user program into core to be executed.

After the user's program has been executed for a period of time, it is placed at the end of the queue (line) of user programs waiting to be run. If only one program is ready to run, it is allowed to do so without interruption until another program is ready. If a user wishes to maintain a permanent copy of his program, he can save a copy within the file area of the disk (an area separate from the swapping area) or on DECtape or paper tape.

1.2 USER FILES

A user is any person logged into EduSystem 50. Each user has an account number and password assigned to him by the System Manager. The account number and password allow the user to gain access to the computer. The account number is also used to identify whatever files the user may own within the EduSystem 50 file system.

The system disk is divided into logical areas called files. A user can store programs or data in files on the system disk. The user can further specify which users may access his files and for what purpose (read, write, or both). Parts of the disk are used to store system files, those programs which are accessible to anyone using

the system. A major portion of this manual deals with how to use the system files, generally called System Library Programs.

With the appropriate Monitor commands, the user can create new files and manipulate old files (extend, reduce, or delete them). These commands are summarized in Table 2-1. Most individual System Library Programs are able to handle user files as input or output with commands issued from the user's console. Such commands are described under the section on the appropriate System Library Program.

1.3 TSS/8 HARDWARE CONFIGURATIONS

A minimum configuration for TSS/8 includes:

a) PDP-8, 8I, or 8E, with at least 12K words of memory and the time-share option. (All are referred to in the following text as PDP-8.) For efficient operation, 16K minimum is highly recommended.

b) RF08 disk with at least one RS08 (can use DF32 with at least two platters, but this is not recommended because of very little area and slow speed).

c) Multi-terminal capability - one or more KL8Es or PT08s or a DC08A.

All, except 8I with DC08A, require a real-time clock.

Optional hardware supported:

a) Up to 32K words of memory.

b) DC08A - may be used only on a PDP-8I and may be used in addition to PT08s.

c) 689AG modem controller - for use with DC08A only.

d) EAE - all instructions of any standard EAE are supported with the exception of the traditional PDP-8 step counter; which is not saved or restored.

e) High-Speed Reader - a paper tape reader is required to build TSS/8. A low-speed reader may be used, however the build procedure will be very time consuming.

f) High-Speed Punch.

g) Line Printer - LP08/LE8 or LS08/LS8E.

h) DECtape - TC01 or TC08 and up to eight drives (NOT TD8E).

i) Up to four disks (three additional RS08s).

j) Card Reader.

k) RK-8E disk (up to four drives).

l) Power failure protection.

m) MI8EF or MI8EG hardware bootstrap.

Software provided with the Standard EduSystem 50 includes the following:

- General-purpose time-sharing Monitor System.
- Time-shared BASIC language processor.
- Time-shared assembly language package including text editor, symbolic assembler, loader, and utility debugging program.
- Time-shared FORTRAN-D and FOCAL language processors.
- System-utility programs, including PUTR, PIP COPY, etc.
- Library of sample programs, textbooks, and curriculum guides.

CHAPTER 2

EDUSYSTEM 50 MONITOR

EduSystem 50 Monitor controls the allocation and use of hardware resources. Many of these functions of the Monitor are invisible, and of no concern to the user; for example, the way it allows many users to run programs on a single computer. In other instances, the user explicitly tells the Monitor what he would like to do by typing one or more of the Monitor commands described in this chapter.

The Monitor commands described in the first half of this chapter are those needed to log into the system, to utilize the System Library Programs, and to log out of the system. All users must be familiar with these commands. The commands described in the last half of this chapter are not needed to run System Library Programs such as BASIC or FOCAL but are frequently useful. The Advanced Monitor commands described in Chapter 10 are primarily useful for creating assembly language programs and files. Table 2-1 contains a summary of the Monitor commands.

TABLE 2-1.

Monitor Commands	Explanation
Logging In and Out	
LOGIN C1 S1	Request to login: C1 = user's account number S1 = user's password
LOGOUT	Request to logout: processing and console time are printed.

Table 2-1. (Cont.)

Monitor Commands	Explanation
TIME C1	Request printout of processing time: C1 = job number If C1 is omitted and the user is logged in, the processing time of the current job is printed. If C1 = Ø, or if the user is not logged in and C1 is omitted the time of day is printed.
Device Allocation	
ASSIGN L1	Reserve device: L1 = R for paper tape reader P for paper tape punch L for line printer C for card reader D for any DECtape unit K for any RKØ5 unit
ASSIGN L1 C1	Reserve specific unit L1 = D for DECtape K for RKØ5 C1 = unit number
RELEASE L1	Release device: L1 = R, P, L, or C (see ASSIGN L1)
RELEASE L1 C1	L1 = D for DECtape K for RKØ5 C1 = unit number
File Handling	
CLOSE S1	Close files: S1 = list of internal file numbers
CREATE S1	Create new file: S1 = name of new file
EXTEND C1 D1	Extend length of file: C1 = internal file number D1 = number of segments to be added to end of file
F C1	Print information about an open file. C1 = internal file number

Table 2-1. (Cont.)

Monitor Commands	Explanation
OPEN C1 S1 C2	Establish association between internal file number and file: C1 = internal file number S1 = file name C2 = account number. If C2 is omitted, the user's account is assumed.
PROTECT C1 C2	Protect a file: C1 = internal file number C2 = new file protection mask (see 10.3.4)
REDUCE C1 D1	Reduce length of file: C1 = internal file number D1 = number of segments to be removed from end of file
RENAME C1 S1	Rename a file: C1 = internal file number S1 = new name of file
Control of User Programs	
DEPOSIT C1 C2...Cn	Store in core memory: C1 = location C2 = contents to be stored in Location C1 C3 = contents to be stored in Location C1+1, etc.
EXAMINE C1 D1	List specified contents: C1 = first location D1 = number of location to be listed D1≤10 decimal
RESTART	Print the program restart address.
RESTART C1	Set program restart address.
START	Restart user program.

Table 2-1. (Cont.)

Monitor Commands	Explanation
START C1	Execute user program: C1 = starting location
Utility Commands	
BREAK	Print current keyboard break mask
BREAK C1	Set keyboard break mask: C1 = new mask
DUPLEX	Echo typed characters on printer.
LOAD C1 S1 LOAD C1 S1 C2 LOAD C1 S1 C2 C3 LOAD C1 S1 C2 C3 C4 LOAD S1 LOAD S1 C2 LOAD S1 C2 C3 LOAD S1 C2 C3 C4	Load core image: C1 = owner's account number; if not specified the user's account is assumed. S1 = name of file C2 = file address of first word to be loaded; if not specified, Ø is assumed. C3 = core address of first word to be loaded; if not specified, Ø is assumed. C4 = core address of last word to be loaded; if not specified, the highest possible value is assumed.
R S1 R S1 C1	Run system file: S1 = name of file C1 = starting address; if omitted, Ø is assumed
RUN S1 RUN C1 S1 RUN S1 C2 RUN C1 S1 C2	RUN USER FILE: S1 = name of file C1 = owner's account number; if omitted the user's account is assumed. C2 = starting address; if omitted, Ø is assumed.
S	STOP execution

Table 2-1. (Cont.)

Monitor Commands	Explanation
SAVE C1 S1 SAVE C1 S1 C2 SAVE C1 S1 C2 C3 SAVE C1 S1 C2 C3 C4 SAVE S1 SAVE S1 C2 SAVE S1 C2 C3 SAVE S1 C2 C3 C4	SAVE core image: C1 = owner's account number; if not specified, the user's account is assumed. S1 = name of file C2 = file address of first word to be saved; if not specified, Ø is assumed. C3 = core address of first word to be saved; if not specified, Ø is assumed. C4 = core address of last word to be saved; if not specified, the highest possible value is assumed.
SWITCH	Print the value of the user's switch register.
SWITCH C1	Set switch register: C1 = word to be set
TALK C1 S1	Send a message to console C1: C1 = destination console S1 = message
UNDUPLEX	Inhibit echo of characters typed to a user program.
USER	Print the user's job number, account number, and console number.
USER C1	Print the job number, account number and console number of job C1.
WHERE	Print the current value of the user's switch register, PC, Link, AC, and EAE registers.

NOTE

All Monitor commands must be terminated by typing the RETURN key. All words within a Monitor command line are separated by one or more spaces.

2.1 CALLING THE MONITOR

The user enters commands to system programs, such as BASIC and FOCAL, in exactly the same way that he enters commands to the Monitor (i.e., by typing them at the keyboard); therefore, the system must have some way of distinguishing between the two cases. It does so by defining two modes of console operation: Monitor mode and program mode. When a user's console is in Monitor mode, all input is interpreted as being commands to the Monitor. Otherwise, all input is assumed to be to the user program or system program which is being run by the user.

A special character, CTRL/B (obtained by pressing B with the CTRL key held down, and echoed on the terminal as ↑B), is used to unconditionally place the user's console in the Monitor mode. Typing CTRL/B tells the system that the command to follow is a Monitor command, regardless of the current console mode. Generally, the command which follows the CTRL/B will be the S command.

↑B	Return to Monitor mode.
↑B↑BS	Return to Monitor mode from a program which is printing. (The two CTRL/B's stop the printout, allowing the S command to be typed.)

It is not necessary to precede each Monitor command with CTRL/B. Once in the Monitor mode, a console stays in that mode until a command is entered to start a system program. To signify that the console is in the Monitor mode, the system prints a dot (.) at the left margin of the console printer paper. This dot indicates that the characters entered next are to be treated as a Monitor command. Thus, the CTRL/B capability is important when a user is running a program and wishes to issue a Monitor command. He may, for example, be using one language (or system program) and want to change to another, as shown below:

NOTE

Characters typed by the user are underlined to eliminate confusion with characters printed by the system.

```
.R FOCAL

SHALL I RETAIN LOG, EXP, ATN ?:NO

SHALL I RETAIN SINE, COSINE ?:NO

PROCEED.

*TYPE 6+10-3-1
=    12.0000*TYPE 25+5*2+5
=    40.0000*↑BS
```

```
•R BASIC

NEW OR OLD--NEW
NEW PROGRAM NAME--
```

Notice that the Monitor responds to CTRL/B followed by S, by printing a dot at the left-hand margin.

2.2 LOGGING INTO EDUSYSTEM 50

To prevent unauthorized usage and to allow the Monitor to maintain a record of system usage, EduSystem 50 requires that each user identify himself to the system before using it. Before attempting to log into the system, the user should ensure that the console LINE/OFF/LOCAL knob is set to LINE and then press the RETURN key. If the console is connected to EduSystem 50 and is not already in use, the Monitor rolls the console paper up two lines and prints a dot at the left margin of the paper. The dot indicates that the system is in Monitor mode and that the Monitor is waiting for a command. The LOGIN command allows the user to gain access to EduSystem 50.

The user types LOGIN followed by an account number and password. Providing the console is free (not already logged in), the command, account number, and password are not printed on the console paper as the keys are typed. If the command name letters are being printed, stop typing the command; instead, strike the RETURN key, log out using the LOGOUT command (see Logging out of EduSystem 50). At this point, a successful LOGIN can be accomplished. The LOGIN command is formatted as shown below:

```
•LOGIN 1234 ABCD     (only the dot appears)
```

The dot (.) is printed by the Monitor, LOGIN is the command name, 1234 represents the user account number, and ABCD represents the password.

NOTE

A command word and each parameter (except the last) is always followed by a space. Command lines are always terminated with the RETURN key. The RETURN key enters the full command line to the system.

When a user types something other than a valid LOGIN command, the Monitor responds in one of the following ways:

. **ILLEGAL REQUEST**	(user typed LOGIN ABCD ABCD)
.LOGIN 4771 DEMO **ALREADY LOGGED IN**	(user typed valid LOGIN on an already logged in console)
. **LOGIN PLEASE**	(user typed ASSIGN D 3)
. **UNAUTHORIZED ACCOUNT**	(user typed an incorrect account number or password)

In the third example, ASSIGN D 3 is a valid command but is not appropriate until the user is logged into the system. In the first example, the Monitor finds that the LOGIN command is improperly formatted (the first parameter must be a 1- to 4- digit number); the console print out tells the user that he has made an ILLEGAL REQUEST. When the console is already logged in and the user types the LOGIN command, the characters typed echo at the console and the Monitor informs the user that the console is occupied with the message ALREADY LOGGED IN.

If the user attempts to use an incorrect account number or password, the Monitor replies UNAUTHORIZED ACCOUNT. Thus the Monitor can distinguish an invalid command from a valid command; it can also distinguish whether the valid command is appropriate when issued,

whether the command is properly formatted, and whether the account number and password are acceptable. In all the preceding examples, Monitor ignores the command and prints another dot.

When the Monitor finds the LOGIN command properly formatted and the account number and password acceptable, it responds by identifying the version of the system being used, the job number assigned to the user, the number of the console being used, and the time-of-day in hours, minutes, and seconds. This information is usually followed by a note from the System Manager concerning the system. For example:

```
.
TSS/8.24    JOB 01   [00,03]  K04     15:26:13

YOU ARE NOW LOGGED INTO THE BHS EDUSYSTEM 50
PROCEED AT YOUR OWN SPEED

.
```

The Monitor then prints another dot and waits for the user to issue the next command. The job number assigned is an internal number by which the system identifies each on-line user; the user need not remember this number.

2.3 LOGGING OUT OF EDUSYSTEM 50

The LOGOUT command indicates to the Monitor that the user is finished and ready to leave his terminal. When Monitor receives a LOGOUT command, it disconnects the user terminal from the system and records the amount of computer time used during the session and the total real time of the session. It also notes any user files deleted or saved. For example:

```
.LOGOUT

JOB  1, USER [12,34] LOGGED OFF K00 AT 19:20:19 ON 27 NOV 74
DELETED  1 FILES (  1. DISK BLOCKS)
SAVED  11 FILES (  38. DISK BLOCKS)
RUNTIME 00:00:24 (  7. CPU UNITS)
ELAPSED TIME 02:14:39
```

Computer processing time used in this example was 24 seconds, while the elapsed time between LOGIN and LOGOUT was 2 hours, 14 minutes, and 39 seconds.

When typing the LOGOUT command, the user may follow it with a colon and an option to initiate some action by the system. To specify an option, the user types, for example:

```
.LOGOUT:K
```

If no option is specified, the S option is assumed; similarly if a user is simultaneously logged in at two (or more) consoles, no files will be deleted until he logs off his last job. For example:

```
.LOGOUT:?

TYPE ↑BS TO ABORT LOG-OUT; OR
TYPE ONE OF THE FOLLOWING (AND CAR RET):

K  TO KILL JOB AND DELETE ALL UNPROTECTED FILES;
L  TO LIST YOUR DISK DIRECTORY;
S  TO SAVE ALL (NON-TEMPORARY) FILES; OR
I  TO INDIVIDUALLY SAVE AND DELETE FILES AS FOLLOWS:

      AFTER EACH FILE NAME IS LISTED, TYPE:
      P  TO SAVE AND PROTECT,
      S  TO SAVE WITHOUT PROTECTING, OR
      CAR RET ONLY TO DELETE.

CONFIRM: L
FIE    .BIN    <17>      1. BLOCKS
BAS000.TMP     <17>      1. BLOCKS
BAS100.TMP     <17>      1. BLOCKS
INTER .BAS     <17>      1. BLOCKS
PROC   .FCL    <12>      2. BLOCKS

CONFIRM: I
FIE    .BIN    <17>      1. BLOCKS  : S
BAS000.TMP     <17>      1. BLOCKS  : DELETED
BAS100.TMP     <17>      1. BLOCKS  : DELETED
INTER .BAS     <17>      1. BLOCKS  :
PROC   .FCL    <12>      2. BLOCKS  : S

JOB  1, USER [ 3,13] LOGGED OFF K00 AT 10:46:07 ON 27 NOV 74
DELETED   3 FILES (   3. DISK BLOCKS)
SAVED  2 FILES (   3. DISK BLOCKS)
RUNTIME 00:00:25 (  2. CPU UNITS)
ELAPSED TIME 00:06:12
```

```
JOB  1, USER (3,13) LOGGED OFF KØØ AT 1Ø:46:Ø7 ON 11-27-74
DELETED    3 FILES (    3. DISK BLOCKS)
SAVED  2 FILES (    3. DISK BLOCKS)
RUNTIME ØØ:ØØ:25 ( 2. CPU UNITS)
ELAPSED TIME ØØ:Ø6:12
```

In the previous example, the user typed a question mark to check the LOGOUT options. When LOGOUT completed the printed explanation, it printed CONFIRM: and waited for a user reply. In this case, the user requested a listing of his files, LOGOUT followed this listing with a second CONFIRM: to which the user replied I. When using the I option, the user is advised not to type his reply to individual entries until printing stops. DELETED is printed automatically by the system to show that the temporary files are deleted without user intervention. The user saved binary file FIE and the FOCAL file PROG. The BASIC file INTER was deleted by typing the RETURN key.

An optional method of logging out of the system is to type K in response to the Monitor dot or K followed by a colon and an option designation. For example:

```
.K

JOB  1, USER [12,34] LOGGED OFF K00 AT 19:20:19 ON 27 NOV 74
SAVED  3 FILES (    4. DISK BLOCKS)
RUNTIME 00:00:07 (  1. CPU UNITS)
ELAPSED TIME 00:13:57
```

A complete list of the logout option can be found in table 2-2 on page 2-13.6

2.4 SYSTEM LIBRARY PROGRAM CONTROL

Once logged into the system, the user can call any EduSystem 50 System Library Program. To call a library program, the user types the command R (meaning run) followed by one or more spaces and the program name. For example:

```
.R BASIC
NEW OR OLD--
```

2.5 TYPING MONITOR COMMANDS

When typing a command to the Monitor, it is possible to correct a letter without retyping the entire line. To do this, press the RUBOUT (or DELETE) key. This will remove characters from the end of the line one by one, and print the deleted characters. When the incorrect characters have been removed, continue typing the correct line. To erase an entire command line and start over, type CTRL/B and then the desired command.

2.6 CONTROLLING OUTPUT

An optional feature of the Monitor allows the user to suspend terminal output from the Monitor. If, for example, he is running a program which causes much output, he may type a CTRL/S (↑S). This signals the Monitor to stop sending to the terminal. When the user wishes to see more output, he types a CTRL/Q (↑Q) , and output continues from where it left off. Users with a VT50 can cause the terminal to sned the ↑S and ↑Q automatically.

2.6.1 System Library Program Control

Once logged into the system, the user can call any EDUsystem 50 System Library Program. To call a library program, the user types the command R (meaning run) followed by one or more spaces and the program name. For example:

```
.R BASIC
NEW OR OLD
```

TABLE 2-2
LOGOUT OPTIONS

Option	Function
:I	Allow the user to individually decide which files to save or delete. Temporary files are deleted automatically.
:K	Delete all unprotected files from disk.
:L	List the user's file directory. After listing the files, the system prints CONFIRM: and the user replies with one of the options.
:Q	LOGOUT without any printed message.
:S	Save all nontemporary files. A temporary file is one of the following: BAS0nn BAS1nn TEMPnn where nn is the console number at which the user is logged into the system. A temporary file is created by a System Library Program and listed in CATALOG listings. A temporary file is also considered to be any file with a .TMP extension. If no option is specified in the LOGOUT command, :S is the default.
:?	Print a listing of the available options and their functions.

The Monitor fetches the BASIC language processor from the System Library and starts it. BASIC begins its dialog by asking if

the user wishes to work on a new program or retrieve an old one from disk storage. Notice that once BASIC begins, the console is no longer in Monitor mode. Dots are no longer printed at the margin. All input is now processed by the BASIC language processor.

If the user types a program name which cannot be found in the System Library, the Monitor responds with an error message and returns the console to the Monitor mode, as follows:

```
.R BASICK
FILE NOT FOUND
```

The exact contents of a System Library may vary from installation to installation. The System Manager may choose to make any number of programs available to all users.

2.7 COMMUNICATION WITH OTHER USERS

Although EduSystem 50 gives each system user the impression that he is the only user of the system, it is actually supporting many users at a time. Often it is useful to communicate with another user, or with the system operator; this is done with the TALK command. The TALK command requests the Monitor to print a message on another system terminal. For example, a user at terminal 7 can ask the system operator to turn on the high-speed punch by typing the following command (the initial dot is printed by the Monitor):

```
.TA Ø PLEASE TURN ON THE HIGH-SPEED PUNCH.
```

The above command causes the following to be printed at console 0.

```
** KØ7  [12,34] **
 PLEASE TURN ON THE HIGH-SPEED PUNCH.
```

KO7 indicates that terminal 7 sent the message. Any terminal can initiate a message to any other terminal. However, if the destination terminal is printing at that time, the message will not be sent. The initiating terminal would, in this case, receive the message BUSY as a response.

2.8 HUNG OUTPUT DEVICES

When the paper tape punch or line printer is off line or hung, the Monitor takes special action so that it is usually possible to continue with little or no data loss. When these devices are hung, the output buffer is not cleared. A system error is generated, and regenerated every few seconds until the condition is cleared. If the user program has not enabled errors, the result will be a printed "HUNG DEVICE" message, and then the terminal bell will ring, trying to persuade the user to do something. There are only two things the user may do to remove himself from this condition. If he is not interested in continuing, he may release the hung device. If he does wish to continue he should put the device on line, and it should take off. He may now type "START" to continue program execution.

2.9 SYSTEM STATUS REPORTS

The command SYSTAT initiates a printout of the full status of EduSystem 50, how many users are on-line, what they are doing, etc. The command SYSTAT is equivalent to typing R SYSTAT. The format of the status report is described in the section on Utility Programs.

The user can obtain information on the amount of computer time used by him, the amount used by another user, or obtain the time of day with the TIME command. The TIME command can be issued in one of the following three forms:

TIME	Returns the elapsed processor time of the user issuing the TIME command since he logged into the system. If the user is not logged in, the time of day is returned.
TIME 0	Returns the time of day.
TIME n	Returns the amount or processor time used by job n since logging into the system.

For example:

```
.TIME
00:00:00
.TIME 0
15:31:18
.TIME 10
00:00:34
```

2.10 RESOURCE SHARING

All system users, when logged into the system, have access to the System Library, disk storage, a virtual 4K PDP-8, and the EduSystem 50 Monitor. The Monitor handles disk resource requests automatically. The Monitor also maintains a pool of available devices which are assigned to users upon request on a first-come, first-served basis. Devices such as the high-speed paper-tape reader cannot, by their very nature, be assigned to several programs simultaneously. Therefore, the Monitor grants individual users exclusive access to these devices when needed. The system disk is not assigned since it can be used by more than one user simultaneously.

All systems include a high-speed paper-tape reader in the pool of available devices. Many systems also include a high-speed paper-tape punch, a high-speed line printer, a card reader, RK8E, and/or one or more DECtapes. These assignable devices are normally used with the System Library Program PUTR to store programs or data for later use.

When a device is assignable (present on the system) and available (not being used), the ASSIGN command may be used to reserve the desired unit or units for exclusive use by the console issuing the command. The valid ASSIGN commands are formatted as shown below:

ASSIGN C	Assign the card reader.
ASSIGN D	Assign a DECtape unit.
ASSIGN K	Assign a RKØ5 unit.
ASSIGN L	Assign the line printer.
ASSIGN P	Assign the high-speed paper-tape punch.
ASSIGN R	Assign the high-speed paper-tape reader.

If other devices are assignable, the System Manager will inform the user of the appropriate device designator. The following is an example of using an invalid device designator:

```
.ASSIGN X
ILLEGAL REQUEST
.
```

The Monitor ignores the request, responds with the appropriate message, and prints another dot. When a valid ASSIGN command is issued, the Monitor checks the availability of the device and responds accordingly. For example:

```
.ASSIGN R
R ASSIGNED
.ASSIGN P
JOB 02  [12,34]  K05  HAS  P
.
```

When the system contains multiple units of a device, the user simply specifies the device; the Monitor assigns an available unit and responds with the unit number. For example:

```
.ASSIGN D
D 0 ASSIGNED
```

If all DECtape units are busy, the Monitor prints the message shown below:

```
.ASSIGN D
DEVICE NOT AVAILABLE
```

A specific unit can be requested by leaving a space between the device designator and the device number. For example:

```
.ASSIGN D 4
D 4 ASSIGNED
```

The ASSIGN command assigns only one device at a time. Therefore, when multiple devices are to be assigned, each must be assigned separately. The following will not accomplish the desired assignments, either with or without the illegal commas.

```
.ASSIGN R , D 2, D 1
R ASSIGNED
```

The Monitor accepted the first device designator and ignored the rest of the command. If device R is unavailable, the Monitor prints the appropriate message. The following commands complete the desired assignments (assuming available devices):

```
.ASSIGN D 2
D 2 ASSIGNED
.ASSIGN D 1
D 1 ASSIGNED
```

When the user has finished working with an assigned device, he should use the RELEASE command to terminate the assignment and allow other users access to the device. (When a user logs out of the system, any devices still assigned to him are automatically released.) A particular device is released when the user enters the RELEASE command, a space, and the device designator (and unit number if required), as shown below:

```
.RELEASE R
.RELEASE D 3
```

In the previous example, the high-speed reader and DECtape unit 3 are released. The Monitor prints a dot on the next line if the release is accomplished; otherwise, it prints a message. If, for example, a request is made to release a device which has not been assigned to the issuing console, the following happens:

```
.RELEASE D 2
ILLEGAL REQUEST
```

The Monitor printed ILLEGAL REQUEST after it checked and found that the specified device was not assigned to the console issuing the command.

NOTE

All commands must be formatted properly; ILLEGAL REQUEST is printed if the user fails to separate the device designator and unit number with a space.

When multiple device units were reserved by a user, each must be individually released. For example:

```
.RELEASE D 1
.RELEASE D 2
.RELEASE R
.
```

The Monitor does not perform checking when releasing a device as it does when assigning a device. The user may have two device units (e.g., two DECtape units) assigned and Monitor would not know which to release; therefore, device numbers are necessary with a RELEASE command. When only one unit of a specific device (one high-speed reader or punch, etc.) is on the system, the device designator alone is sufficient.

2.11 ERROR MESSAGES

An appropriate error message is printed whenever: a Monitor command cannot be performed at the time it is requested, a typing error is made, or the command is illegal (or nonexistent). Following each error message, the Monitor ignores the command and prints another dot, after which the user can issue another command. Table 2-3 is a list of the Monitor error messages.

TABLE 2-3
MONITOR ERROR MESSAGES

Message	Explanation
ALREADY LOGGED IN	The user tried to log in on a console which is already in use.
BUSY	The user attempted to talk to a console which is currently printing or on which another user is typing.
DEVICE NOT AVAILABLE	An ASSIGN command has been entered for either: 1) a device which is not present on the system, or 2) a device which is temporarily busy and should be available in a few seconds.
FULL	The system is full. Another user cannot log in until one of the present users logs out.

Table 2-3. (Cont.)

Message	Explanation
ILLEGAL REQUEST	The user requested an illegal command. This error usually results when some parameter has been given an incorrect value or the request refers to a facility not owned by the user.
LOGIN PLEASE	The user attempted to use a console which is not logged into the system.
S1??	The System Interpreter does not understand the command. S1=command.
TYPE ↑BS FIRST	The user attempted to use a system command which cannot presently be honored due to the status of the user's program.
UNAUTHORIZED ACCOUNT	The user attempted to log into the system with an invalid account number or password.
WAIT FOR I/O	A request cannot be honored because a device is busy. Try typing the command again.

CHAPTER 3

SYSTEM LIBRARY PROGRAMS

The System Library contains a comprehensive set of user programs for a wide range of applications. Language processors, such as BASIC and FOCAL, allow the user to code and run programs in interactive languages. FORT compiles and executes programs written in FORTRAN language. A complete assembly language system allows programs to be written in PALD, assembled, and run. Various utility programs perform special functions. The DEC-supplied System Library consists of the following programs.

- BASIC - an easily learned interactive language originally developed at Dartmouth College.
- CAT - used to list all the files which a user has stored in his library.
- COPY - a utility program used to transfer files between the system disk and DECtape.
- EDIT - a line-oriented text editor, used to create and modify source programs (such as FORTRAN) and data files.
- FOCAL - DEC's own interactive language for on-line problem solving, designed especially for use on mini-computers.
- FORT - a modified version of FORTRAN II.
- LOADER - a binary loader used to load assembled programs for execution.
- ODTHI - Octal Debugging Technique for testing and modifying assembly language programs.
- PALD - a 2-pass symbolic assembler.
- PIP - Peripheral Interchange Program for transferring files between the system disk, paper tape, and line printer.
- PUTR - a utility program used to transfer files between all EduSystem 50 devices.
- SYSTAT - (System Status) a utility program that prints a brief description of the system status.

A more detailed description of each of the above System Library Programs is presented in the following sections.

3.1 GENERAL FILE CHARACTERISTICS

A fundamental feature of the Monitor is its ability to save programs or other data for each user in his own private library. These individual user libraries are maintained on the system disk. Individual entries in the library are called files, whether they contain programs or data. Within the library itself, there is no distinction between types of files by their contents. Each file is identified with a file name by which it is known and called into use.

The user does not directly create and update the files in his library. He uses the System Library Programs for this purpose. For example, he can use the SAVE command in BASIC. The SAVE command takes the BASIC program named and saves it as a file in the user's library for future use. Similarly, EDIT can be used to modify an existing file, resulting in the creation of a new file. Therefore, although the Monitor provides the actual file storage capability, most file manipulation is done while System Library Programs are being run.

The System Library Programs which operate on these files must know which file to use, when to create a new file, and what to call it. Each Library Program has its own method of determining whether a user wishes to use an old file or create a new one; this is explained in the sections on individual library programs.

Example 1:

```
•R BASIC

NEW OR OLD--OLD
OLD PROGRAM NAME--PRIME

READY
```

Example 2:

```
.R FORT

 INPUT:TYPE
OUTPUT:BTYPE
```

For most of his work, the user requires access to only his own library. However, it is often a useful feature to be able to obtain a program from another user's library, allowing a single file to be shared by several users. To access a program from another user's library, the user must tell the system in which individual library the file is stored. The user tells the system by entering the account number of the library's owner. (In the absence of an account number, the user's own library is the assumed source.) To get a file from the System Library, type an asterisk immediately after the file name.

Example 1:

```
.R BASIC

NEW OR OLD--OLD
OLD PROGRAM NAME--HOSSR*

READY


.R PALD

 INPUT:NOTPIP 5440
OUTPUT:BIN1
```

NOTE

> Most examples in the discussions of individual System Library Programs use file names within the user's own library. The user is free (file protect permitting) to use files from other user's libraries.

Access to another user's files is gained only with his permission. A user may "protect" his files against other users, i.e., prevent them from gaining access to his files, even though they know his program

name and account number. Library Programs never permit a user to write in another user's files. Specifying a file which is protected, or specifying a nonexistent file, is an error that is detected immediately. An error message is printed and the file name is requested again.

The user places his output in a single file; however, it is often useful to input several files together. (For example, the user may wish to assemble two parts of a PAL-D program together.) To specify more than one input file, separate the file names by commas. No Library Program (except PUTR) allows more than three input files. FORTRAN is limited to two; BASIC allows only one.

BASIC is a self-contained programming system, with an editor, compiler, and run-time system. It also has a distinctive file format. Files created by BASIC are not compatible with files created by other Library Programs. All other Library Programs depend on each other; therefore, all other Library Programs use the same format for their disk files. Consequently, files created by the Editor can be used as input to PAL-D or FORTRAN programs as data files.

The Monitor includes a disk quota system, which limits the amount of file space that each account can have. A user can find out what his own quota is by running the program CAT. Any time an account exceeds this disk quota, the Monitor will print a message such as [MYFILE EXCEEDING DISK QUOTA] warning the user that he is about to run out of disk space. The amount by which he may exceed his disk quota is called the grace quota, and is defined by the System Manager.

Up to this point, only files that exist within the time-sharing system, i.e., on the system disk, have been described; however, EduSystem 50 provides other means of file storage such as paper tape, RKØ5 cartridge disk, and DECtape. The Library Programs PIP, COPY, and PUTR can be used to transfer data between any devices which are present on the System.

3.2 CONTROLLING THE EXECUTION OF SYSTEM LIBRARY PROGRAMS

EduSystem 50 provides the user with two options for stopping the system. CTRL/C (C with the CTRL key held down) allows the user to stop his BASIC program and return to the beginning of that program without returning to the Monitor. For example, if the user begins to run a BASIC program that has an endless loop, he can type CTRL/C to stop it. BASIC responds to ↑C with READY. All other Library Programs respond in a similar manner.

CTRL/B is used to stop the Library Program most recently called. CTRL/B followed by S and the RETURN key unconditionally returns the user to the Monitor mode; the user can then call another Library Program. If the system is printing, two CTRL/B's and the S (↑B↑BS) are required to stop the system.

RUBOUT is another useful character that deletes the last typed character. Some Library Programs respond by printing \ or ← while others print the deleted character. If the RUBOUT key is typed while entering file names for input or output to a Library Program, RUBOUT deletes the whole line. The request for input or output is then repeated.

3.3 RETURNING TO THE MONITOR

The user can stop the execution of a System Library Program at any time by typing CTRL/B followed by S and the RETURN key. The System Library Programs can also initiate a return to the Monitor. When the System Library Programs initiate a return, ↑BS is printed just as though the user had terminated the program. For example, BASIC returns to the Monitor when the user types the BYE command:

```
READY

BYE
↑BS
.
```

FORTRAN returns to the Monitor after completing execution of a program. CAT and SYSTAT return after printing their particular data output. PAL-D returns after completion of an assembly, LOADER at the end of a normal load, and EDIT after completion of an EDIT. FOCAL, ODT, and PIP never return to the Monitor; these programs must be terminated by the user with CTRL/B followed by S. Some System Library Programs return to the Monitor when a fatal error condition is detected.

CHAPTER 4

CALLING AND USING BASIC

4.1 BASIC

EduSystem 50 BASIC is a time-sharing version of the BASIC language. It allows even the beginning computer user to write and run meaningful programs. In addition, EduSystem 50 BASIC has advanced language features such as strings, files, and program chaining. This section describes the BASIC language capabilities not discussed in Chapter 1 of the EduSystem Handbook. Table 4-4 contains a complete summary of the EduSystem 50 BASIC language.

To call BASIC, the user types:

```
•R BASIC
```

After the user logs into EduSystem 50, and calls BASIC in the above manner, BASIC prints NEW OR OLD--. The user then types the appropriate adjective: NEW (if he wants to enter a new program) or OLD (if he wants to retrieve a program that was previously filed).

BASIC then asks NEW PROGRAM NAME-- (or OLD PROGRAM NAME--) and the user types any combination of six letters or less. If the user is recalling an old program file from the disk, he must use exactly the same name as when he originally instructed BASIC to save it.

BASIC prints READY to signal the start of the editing phase; the user then begins to type the new program. If the user types a line consisting of only a line number followed by the RETURN key, that line is deleted. Each line must begin with a line number greater than 0 and less than 2047 and which contains no non-digit

characters. To enter an entire line to the computer, the user must press the RETURN key.

If the user makes a typing error while typing a statement and notices it immediately, he can correct it by typing the RUBOUT (or DELETE) key (right-hand side of the keyboard), or the back arrow key. Typing either key deletes the character in the preceding space and prints a backarrow (←) character for each character erased. The user can then type the correct characters. Typing the RUBOUT key a number of times erases one character from the current line (spaces are characters) to the left for each RUBOUT typed.

While BASIC is in the editing phase, certain additional commands (which must not have line numbers) are available. The commands are described in Table 4-4 under Edit/Control Commands.

4.2 LANGUAGE FEATURES

4.2.1 Truncation Function, FIX(X)

The truncation function returns the integer part of X. For example:

```
10 PRINT "FIX(10.2)=" FIX(10.2)
20 END
RUN

FIX(10.2)= 10
```

FIX is like INT for positive arguments, and can be defined as:

```
FIX(X) = SGN(X) * INT(ABS(X))
```

4.2.2 ON GOTO Statement

The ON...GOTO statement may be used to provide a many-way branch. The general form of the ON...GOTO is:

ON expression GOTO line number, line number ...

If the value of the integer part of the expression is 1, a GOTO is performed to the first statement. If the value of the integer part of the expression is 2, a GOTO to the second statement number is performed, etc. If the value is less than one, or greater than the number of statement numbers, the program terminates and an error message is printed. Examples of ON GOTO are shown below:

```
999 ON N GOTO 100,400,200,600,499

872 ON A+SQR(B*C) GOTO 100,200
```

4.2.3 SLEEP Statement

The SLEEP statement causes a BASIC program to pause for a specified interval, then continue running. SLEEP is followed by the number of seconds the program is to pause. For example:

```
222 SLEEP 30
```

or

```
220 LET N=15
222 SLEEP 2*N
```

causes a 30 second delay in the program.

The SLEEP statement is a useful way for a program to wait for a device (DECtape or line printer) which is busy. The ELSE clause in the OPEN statement can go to a routine which pauses for a while, then retries the OPEN. When the current user finishes with the device and releases it, the program may then proceed to OPEN and use it. This capability is especially useful when many users may be looking up information on a single DECtape file. It may also be used to allow two programs to communicate with each other. Each writes information on a tape file for the other, or others, to read.

SLEEP should always be used when waiting for a device. While the program is sleeping it is not using any processor time. A SLEEP time of 30 to 60 seconds is recommended. It is particularly important that the program not wait by repetitively retrying the OPEN. To do so wastes computer time and slows down other users. The integer part of the argument is used to determine the number of seconds to delay. This value must be between 0 and 4095.

4.2.4 Comments

An entire statement of comments may be included in the BASIC program by means of the REM statement. Often comments are easier to read if they are placed on the same line with an executable statement rather than in a separate REMARK statement. This can be accomplished by ending an executable statement with an apostrophe. Everything to the right of the apostrophe up to the statement terminator (carriage return or backslash) is ignored (unless the apostrophe occurs within a print literal or string constant.) For example:

```
10 LET X=Y 'THIS IS A COMMENT'
20 PRINT "BUT THIS IS NOT A COMMENT"
30 LET X$="A'B"
```

Thus, a comment is added to line 10 with an apostrophe, but in lines 20 and 30 the apostrophe is treated as a valid character.

4.2.5 Blank Lines

To make BASIC programs easier to read, blank lines can be inserted anywhere in a BASIC program. These can be used to break a program into logical sections, or (as is often done) to insert remarks with the apostrophe feature. For example:

```
10 'PROGRAM WRITTEN BY SAM JONEX-S
100
```

Note that to insert a blank line, you must type one or more spaces after the line number; typing the line number alone will just delete that line from the program.

4.2.6 Multiple Statements Per Line

As many statements as will fit may be typed on a single program line. Each statement must be separated by the backslash character "\". The only statement requiring a line number is the initial one. For example:

```
10 FOR I=1 TO 10\PRINT I\NEXT I
```

Note that the backslash character acts as a statement terminator and thus cannot be included in a comment statement.

4.2.7 Editing BASIC Statements

If a program line is incorrect, it can be corrected by retyping it. Minor errors in statements can be corrected by using the EDIT command. The user types EDIT followed by the line number of the statement to be edited. BASIC responds by printing a left bracket ([). The user then types a search character. BASIC prints a close bracket and prints the statement through the first occurrence of the specified search character. The user may then:

1. Type new characters which are inserted at that point in the statement.
2. Type one or more back arrows (←) to delete characters to the left of the search character.
3. Type the ALT MODE key to delete the entire line up to that point (but not the line number).
4. Type CTRL/L to continue to the next occurrence of the search character.
5. Type CTRL/G to specify a new search character.

6. Type LINE FEED to finish the edit, keeping the remainder of the line unchanged.
7. Type RETURN to finish the edit, deleting the remainder of the line.

4.2.8 Saving Compiled Programs

BASIC compiles the current program each time it is run. If, however, a program will be used frequently without being changed, it may be stored in its compiled form. A compiled program can be retrieved and executed faster than a BASIC source program. To save a compiled program, the user types, for example:

```
COMPILE FAME
```

The program is saved on the disk under the specified name (FAME, in this case). If a file by that name exists, BASIC prints DUPLICATE FILE NAME and does not compile that program.

Once a program has been compiled, it may be retrieved and run just like an ordinary BASIC source program. It may not, however, be listed, saved, or changed. If an attempt is made to do any of these things, the message EXECUTE ONLY is printed. The compile capability may therefore be used to protect programs from unauthorized listing or changing. Since only BASIC source programs can be edited, the user may wish to store both a source and a compiled version of a given program.

Compiled files are distinguished from regular BASIC programs by their file extensions. BASIC source programs have an extension of .BAS. Compiled files have an extension of .BAC. These extensions are printed along with the file name when a catalog is requested.

4.2.9 File Protection

EduSystem 50 permits a user to specify a protection code for each file. (See Chapter 10 for a full description of protection codes.) The commands which write disk files (SAVE, REPLACE, COMPILE) also permit the user to specify what protection is to be given to a file. This is done by following the file name with the protection code in angle brackets. For example:

```
SAVE DEMO <10>
```

will create and save a file named DEMO.BAS having a protection code of 10. When no protection is specified, a protection of 12 is automatically assumed.

4.2.10 Project-Programmer Numbers

In specifying the Account Number prior to requesting an OLD file, the user may optionally type a Project-Programmer number (giving the Account Number as two 2-digit numbers separated by commas instead of a single 4-digit number). In this way, the user may RUN files from another user's disk area. For example, both of the following are acceptable:

```
OLD PROGRAM NAME--FILE 13,3
```

where 13 is the Project Number and 3 is the Programmer Number, or:

```
OLD PROGRAM NAME--FILE 1303
```

where 1303 is the account number. The two file name indications are equivalent.

4.2.11 Restricted Accounts

As an added system protection, BASIC checks to see if an attempt is being made to run BASIC under Accounts 1 or 2. If so, BASIC prints the error message:

```
IMPROPER ACCOUNT #

ABORT
↑BS
```

thus preventing BASIC from interfering with the System Directories or the System Library.

4.2.12 Catalog Format

The CATALOG command prints the user's directory, file names and file extensions, file size, and file protection codes. For example:

```
CATALOG

NAME        SIZE PROT
TEMP00         1   12
DEMO  .BAS     1   10
IBOLD .BAC     1   10
BAS000.TMP     1   17
BAS100.TMP     1   17
```

4.2.13 Strings in BASIC

EduSystem 50 BASIC has the ability to manipulate alphabetic information (or strings). A string is a sequence of characters, each of which is a printing ASCII character (see Appendix B). EduSystem 50 strings consist of one to six characters; strings of more than six characters are truncated on input to six characters.

Variables can be introduced for simple strings, string arrays, and string matrices. A string variable is denoted by following the variable name with the dollar sign character ($). For example:

A1$	A simple string of up to six characters.
V$(7)	The seventh string in the array V$(n).
M$(1,1)	An element of a string matrix M$(n,m).

When string arrays or matrices are used, a DIM statement is required. For example:

```
10 DIM V$(10),M$(5,5)
```

reserves space for eleven 6-character strings for the array V$, and space for 36 6-character strings for the matrix M$.

4.2.13.1 Reading String Data -- Strings of characters may be read into string variables from DATA statements. Each string data element is a string of one to six characters enclosed in quotation marks. The quotation marks are not part of the actual string. For example:

```
10 READ A$,B$,C$
200 DATA "JONES","SMITH","HOWE"
```

The string JONES is read into A$, SMITH into B$, and HOWE into C$. If the string contains more than six characters, the excess characters are ignored. The following program:

```
10 READ A$
20 PRINT A$
30 DATA "TIME-SHARING"
40 END
RUN
```

causes only

```
TIME-S
```

to be printed.

String and numeric elements may be intermixed in DATA statements. A READ operation always fetches the next element of the appropriate type. In the following example:

```
10 READ A,A$,B
20 DATA "YES",2.5,"NO",1
```

2.5 is read into A, YES into A$, and 1 into B.

The standard RESTORE statement (as described in Chapter 4) resets the data pointers for both string and numeric elements. Two special forms of the RESTORE command, RESTORE* and RESTORE$, may be used to reset just the numeric or string data list pointers, respectively. For example:

```
10 READ A,A$,B
20 DATA "YES",2.5,"NO",1
30 PRINT A,A$,B
40 RESTORE*
50 READ A,A$,B
60 PRINT A,A$,B
70 END
RUN
```

would print:

```
 2.5          YES          1
 2.5          NO           1
```

If line 40 were changed to RESTORE, this program would print:

```
 2.5          YES          1
 2.5          YES          1
```

since the numeric as well as string data lists would be reset.

4.2.13.2 Printing Strings -- The BASIC PRINT statement may be used to print string information. If the semicolon character is used to separate string variables in a PRINT command, the strings are printed with no intervening spaces. For example, the program:

```
10 READ A$,B$,C$
20 PRINT C$;B$;A$
30 DATA "ING","SHAR","TIME-"
40 END
RUN
```

causes the following to be printed:

```
TIME-SHARING
```

4.2.13.3 Inputting Strings -- String information may be entered into a BASIC program by means of the INPUT command. Strings typed at the keyboard may contain any of the standard ASCII characters on the user terminal except back arrow (←) and quotation mark ("). Back arrow is used in BASIC to delete the last character typed. Commas are used as terminators just as with numeric input. If a string contains a comma, the entire string must be enclosed in quotation marks. The following program demonstrates string input.

```
10 INPUT A$,B$,C$
20 PRINT C$,B$,A$
30 END
RUN

? JONES,SMITH,HOWE
HOWE            SMITH           JONES

READY
```

Strings and numeric information may be combined in the same INPUT statement as in the following example. Note that if an input string contains more than six characters, only the first six are retained.

```
10 INPUT A,A$,B$
20 PRINT A$,B$,A
30 END
RUN

? 01754,MAYNARD,MASS.
MAYNAR          MASS.           1754
```

The numeric variable A is set to 1754 (leading zeros are deleted), the string MAYNAR is put in the string variable A$, and the string MASS. is put into the string variable B$. To print the number 01754, the number could be input and output as a character string.

4.2.13.4 Line Input -- Strings of more than six characters may be entered by means of the LINPUT (line input) statement. A LINPUT statement is followed by one or more string variables. For example:

```
10 LINPUT A$(1),A$(2),A$(3),A$(4),A$(5)
```

The first six characters to by typed are stored in the first string variable, the next six in the second, and so until the line of input is terminated by a carriage return.

Commas and quotes are treated as ordinary characters and hence are stored in the string variables. For example, if the following line were typed in response to the above LINPUT command:

```
?MAYNARD, MASS. 01754
```

then the values of the string variables would be as follows:

```
A$(1) = "MAYNAR"
A$(2) = "D, MAS"
A$(3) = "S. 017"
A$(4) = "54"
A$(5) = ""1
```

[1]Strings may consist of zero characters. Such a string is empty (or null). If printed, it causes nothing to be output. The null string is usually represented by a pair of quotes with nothing between (""). The null string should not be confused with a string of one or more spaces.

In the above example, the maximum number of characters which could be typed would be 30. Any additional characters would be ignored. In all cases, the maximum number of characters which may be typed in response to LINPUT is 50. If a longer line is typed, the message LINE TOO LONG is printed. The input line is ignored and must be reentered.

It is possible to mix numeric and string variables in a LINPUT statement, but this practice is not recommended. As an illustration of how this might be done, consider the example given earlier:

```
10 LINPUT A,A$,B$
```

where the user might type:

```
? 01754,MAYNARD, MA
```

This still sets the numeric variable A to 1754 (when used in LINPUT statements, numeric input remains unchanged). However, the string variable A$ would now be MAYNAR, and the string variable B$ would be D, MA.

When inputting strings with LINPUT, the error messages: MORE? and TOO MUCH INPUT, EXCESS IGNORED cannot occur.

4.2.13.5 Working With Strings -- Strings may be used in both LET and IF statements. For example:

```
10 LET Y$="YES"
20 IF Z$="NO" THEN 100
```

The first statement stores the string YES in the string variable Y$. The second branches to statement 100 if Z$ contains the string NO. For two strings to be equal, they must contain the same characters in the same order and be the same length. In particular, trailing

blanks are significant since they change the length of the string. "YES" is not equal to "YES ".

The relational operators < and > may also be used with string variables. When used with strings, these operators mean "earlier in alphabetic order" or "later in alphabetic order", respectively. They may be used to alphabetize a list of strings, for example. The relation operators >=, <=, and <> may be used in a similar manner. The arithmetic operations (+, -, *, /, ↑) are not defined for strings. Thus statements such as LET A$ = 3*5 and LET C$ = A$+B$ have no meaning, and should not be used in a BASIC program. They will not cause a diagnostic to be printed; however, the results of such operations are undefined.

4.2.13.6 The CHANGE Statement -- The CHANGE statement may be used to access and alter individual characters within a string. Every string character has a numeric ASCII code (see Appendix B), a number which is used to indicate that particular character. The CHANGE statement converts a string into an array of numbers, or vice versa. The CHANGE statement has the form:

```
100 CHANGE A TO A$
```

or

```
100 CHANGE A$ TO A
```

where A$ is any string variable (or an element of a subscripted string variable) and A is an array variable with at least six elements. Any array variables used in CHANGE statements must have appeared in a DIM statement with a dimension of at least six.

The following program illustrates the use of the CHANGE statement by changing a string variable into an array of numbers.

```
10 DIM A(6)
20 READ A$
30 CHANGE A$ TO A
40 PRINT A(0);A(1);A(2);A(3);A(4);A(5);A(6)
50 DATA "ABCD"
60 END
RUN

 4  65  66  67  68  0  0
```

The CHANGE statement takes each character of the string and stores its corresponding numeric (ASCII) code in elements one to six of the array. Remaining array elements are set to zero. The length of the string (0-6 characters) is stored in the zero element of the array. In the example above, the character codes for A, B, C, and D are stored in A(1) to A(4). A(5) and A(6) are set to zero. The number 4 is stored in A(0) since the string A$ is four characters long.

CHANGE may also be used to change an array of numeric codes into a character string as in the following program:

```
10 DIM A(6)
20 FOR I=0 TO 5
30 READ A(I)
40 NEXT I
50 CHANGE A TO A$
60 PRINT A$
70 DATA 5,69,68,85,53,48
80 END
RUN

EDU50
```

The length of the resulting string is determined by the zero element of the array. In the previous example, the string is five characters long. The elements of the array, starting at subscript 1, are assumed to be numeric character codes; these are converted to characters and are stored in the string. If any codes encountered are not valid character codes, or if an invalid string length is given, the message BAD VALUE IN CHANGE STATEMENT AT LINE n is printed, and execution is stopped.

A BASIC string of less than six characters always has the remaining character positions filled with zeros. For this reason, when such a string is changed to an array, the first six array elements are set to zero. The CHANGE statement always fills six array elements, even though the strings may not be six characters long. The user should be careful to dimension the array used in a CHANGE statement to at least six. If a string of characters is transformed into an array of less than six elements, an undetected error will occur.

The CHANGE statement is usable with strings not created by BASIC. It may, for example, be used to access files other than BASIC data files. Each string variable corresponds to three PDP-8 words. The CHANGE statement treats these three words as six 6-bit bytes, converts each 6-bit byte to its numeric character code equivalent and stores it in the corresponding array element. The zero element of the array, the string length, is set equal to the number of bytes (characters) before the first zero byte. When reading unspecified data, there may be non-zero bytes following this zero byte. If so, they will be transferred to the array as well.

4.2.13.7 The CHR$ Function -- Occasionally, it is desirable to type a character other than those in the printing ASCII set, or to compute the value of a character to be printed. For this purpose, the CHR$ function can be used in a PRINT statement. The argument of the CHR$ function is sent as an ASCII character to the Teletype. For example:

```
10 FOR I=0 TO 9
20 PRINT CHR$(I+48);
30 NEXT I
40 END
```

prints 0123456789, since 48 to 57 are the ASCII values for the characters 0 to 9. The following special characters can also be printed using the CHR$ function:

Bell	CHR$(7)
Line feed	CHR$(10)
Carriage return	CHR$(13)
Quote (")	CHR$(34)
Back arrow (←)	CHR$(95)
Form feed	CHR$(12)

The Teletype will accept characters from 0 to 255 (decimal), many of which do nothing on most kinds of teletypes. Some of the special (non-printing) characters should not be used. For example, CHR$(4) causes a Dataphone to disconnect.

For each ASCII code there is a second acceptable form permitted in CHANGE and CHR$. The second code is obtained by adding 128 to the code given in the table in Appendix B. For example, CHR$ would type A in response to either 65 or 193 as an argument.

4.2.14 Program Chaining

Most programs are easily accommodated by EduSystem 50 BASIC. If a program becomes very long, however, it may be necessary to break it into several segments. Typically, programs of more than two to three hundred statements must be split into more than one file. A program that has been broken into more than one piece is commonly called a chained program.

Each part of a chained program is saved on the disk as a separate file. The last statement of each part to be executed is a CHAIN statement specifying the name of the next part of the program. The next file is then loaded and executed. It may in turn chain to still another part of the program. The general form of the chain command is:

```
414 CHAIN "NAME"
```

or

```
414 CHAIN A$
```

where NAME is the name of the next segment to be executed (one to six characters enclosed in quotation marks). The name of the next segment may also be contained in a string variable. In either case, the file of that name is loaded and run. Thus, the statement:

```
999 CHAIN "SEG2"
```

is equivalent to:

```
OLD
OLD PROGRAM NAME--SEG2

READY

RUN
```

except that it happens automatically. Each separate part of the program automatically links to the next part of the program chain.

The individual sections of a chained program may be either regular source files (.BAS) or compiled files (.BAC). If the sections are source files, they must be compiled before they are run. A chained program runs more efficiently if all its sections have been compiled. Source and compiled files cannot be mixed in program files.

If an error occurs while compiling or running a chained program, the name of the section containing the error is printed as part of the error message. In all cases, whether a program terminates by an error or a STOP or END, BASIC returns to the first program in the chain. This is the one which is available for editing and rerunning when BASIC prints READY.

Most chained programs require that information from one section be passed to the next. The first section may, for example, accept input values and perform some preliminary calculations. The intermediate results must then be passed to the next section of the programs. This passing of values is done by means of data files which are explained in the next section. Whenever a CHAIN operation is performed, program data which has not been saved in a file is lost.

Variable and array values are not automatically passed to the next program.

4.3 DISK DATA FILES

The standard BASIC language provides two ways of handling program data items. They may be stored within the program (in DATA statements) or they may be typed from the terminal. DATA statements, however, allow for only a limited amount of data. Also, the data is accessible only to the program in which it is embedded. Typing data from the terminal allows it to be entered into any program, but this is a time-consuming process. In either case, the data or results of calculations cannot be conveniently stored for future use. All these limitations may be overcome by the use of disk data files.

A data file is separate from the program or programs which use it. It is a file on the disk similar to a saved program, but it contains numbers or strings rather than program statements. This information may be read or written by a BASIC program. (Information in a data file is stored in a coded format; therefore, it cannot be listed by the BASIC Editor or EDIT.) (The maximum size of a data file is about 350,000 characters.) String and numeric information may be combined in a single data file. The number of data files a user may have is limited to about 100, space allowing. When a file is first created, its contents are undefined.

4.3.1 File Records

A data file is made up of logical units called records. A record may be as small as a single numeric or string variable. More typically, it is a group of variables or arrays. The design of the program usually dictates the most efficient size of the record. If, for example, the program manipulates a series of 5 by 5 matrices, each record could contain one such matrix. If the program operates on 80-character alphanumeric records, 14 string variables might comprise a record.

The size and composition of a record are defined with a RECORD statement. Like the DIM statement, RECORD is followed by a series of variables. They may, however, be unsubscripted as well as subscripted. For example:

```
10 RECORD A(5,5)
10 RECORD B$(14)
10 RECORD A,B,C$(8),D,E(5)
```

The set of variables mentioned in a RECORD statement, taken together, constitute a record. Each element within the record is a field. Numeric and string information may be mixed to comprise a more convenient record.

Variables mentioned in a RECORD statement should not appear in a DIM statement. The RECORD statement reserves variable space exactly as a DIM statement does. The difference is that the variables are also identified as being used for file input and output. Non-subscripted variables appearing in RECORD statements must not have been used previously in a program; therefore RECORD statements should always be the first statements in a program.

Records may be any length. A long record is typically more efficient since more information is transferred in a single operation. Records should, however, be only as long as necessary since excess variables lengthen the file. In particular, it is important to remember that all arrays and matrices have zero elements. The array A(5,5) has 36 elements, not 25. If A appears as part of a record, all 36 elements should be used.

It is also useful to try to make record sizes 43 variables long, or a multiple of 43. Each RECORD statement reserves program variable space in units of 43 whether or not the record is that big. Unless the record fills this area, some program variable space is wasted. It is not worthwhile, however, to make an inherently small record 43 variables long just to conform to this convention; this would make the file unnecessarily large.

4.13.2 Opening a Disk File

Disk data files are completely separate from the programs which use them. Therefore, the program must specify which file or files it will use. The OPEN command is used for this purpose. OPENing a disk data file associates it with an internal file number, either 8 or 9. (A program may have two disk data files open at one time.) For example:

```
100 OPEN 9, "DATA10"
100 OPEN 8,A$
```

The name of the file to be opened may be explicitly stated in the OPEN command. If it is, it must be contained in quotation marks. The file name may also be contained in a string variable, allowing the program to decide which file to open, perhaps on the basis of input from the program's user. In either case, the name of the file is preceded by the internal file number, either 8 or 9. This argument may also be an expression whose value is either 8 or 9.

When a file is opened on an internal file number which has a file already open, the previously opened file is closed and the new file opened.

If no file of that name exists, the file is created. In either case, once the file is open, it is available for both reading and writing. BASIC disk data files are assigned an extension of .DAT which need not be specified as part of the file name in the OPEN statement.

4.13.3 Reading/Writing Disk Files

Once open, files may be read and written, one record at a time, using the GET and PUT statements. GET statements read one record of information directly into the variable in the RECORD statement.

PUT statements write the present values of the variables in the RECORD statement. Both GET and PUT statements are followed by the internal file number (8 or 9 or an expression), the line number of the RECORD statement containing the variables to be transferred, and the name of a control variable. For example:

```
100 RECORD A,B,C$(30),D(8)
110 OPEN 8,"FILE1"
120 LET I=0
130 GET 8,100,I
```

The control variable specifies the file record to be transferred. In the example above, FILE1 is opened as internal file 8. The value of the control variable, I, is zero. The GET statement in line 130 reads the first record (record 0) of FILE1 into A, B, and the arrays C$ and D. Single numeric values are read into A and B. 31 strings are read in C$, and 9 numeric values are read into D. After each transfer, whether it is a GET or a PUT, the value of the control variable is automatically incremented. Successive GET's or PUT's automatically proceed to the next record of the file.

The PUT statement has a similar format. For example, if line 130 of the preceding program had been:

```
130 PUT 8,100,I
```

the present values of A, B, C$ and D would have been written to the first record of FILE1.

File records may be accessed randomly by setting the control variable to the desired record number before doing the GET or PUT. Single records may be read, changed, and then written without processing the entire file. When reading a file, the record referenced in the GET statement must, of course, be the same as the record referenced in the PUT statement which wrote the data onto the file. The total length of the record and the relationship of string and

numeric fields within the records used for the GET's and PUT's must be the same. If they are not, improper information will be read and written.

New files may be created by opening a file which does not already exist. As successive records are written onto the file, its length is extended as necessary. When a new file is created, it is useful to immediately write an end-of-file code in the last record. Writing the last record first forces the entire file to be allocated, making sure that enough disk space is available. It also provides an end-of-file mark. Programs which read this file may then check for this end-of-file mark to avoid reading past the end of the data file which results in an error.

Existing files may be enlarged by writing a new record farther out. If the program does not know how big the file will be, it may simply write records to the file in sequence. The file will be automatically extended. When all the records have been written, one final end-of-file mark can be added.

In general, all records read or written on a specific file should be the same length, i.e., contain the same number of variables. However, if the user is careful he may intermix records of different lengths in a file. Suppose the following statement is executed:

```
40 PUT 8,100,N
```

and the value of N is n and the record specified by statement 100 is of length m. The PUT statement will write m variables in the file starting at the m*n variable. The simple rule for computing the first variable in the file to be accessed is the record length times the record number. (Remember the first record is record number zero.)

4.3.4 Closing/Deleting Disk Files

When all work has been completed on a data file, it should be closed with a CLOSE statement. Once the file is closed, it may not be read or written unless it is reopened. The file does, however, remain on the disk and is available for future use. The CLOSE statement is followed by the internal file number to be closed (8 or 9). For example:

```
950 CLOSE 8
```

If the disk file was just created for temporary scratch use (to pass parameters during a CHAIN, for example), it should be deleted at the end of the program instead of closed. The UNSAVE statement is used to delete files. For example:

```
1000 UNSAVE 9
```

The file opened on internal file number 9 is deleted from the disk. Both CLOSE and UNSAVE may be followed by an expression equating to 8 or 9 instead of a constant.

Open disk data files are automatically closed at the end of the program, unless the program CHAINs to another program. In this case, all open files remain open and the new program may access them without executing an OPEN statement.

4.4 DECTAPE DATA FILES

Large permanent data files are best stored on DECtape rather than on disk. Each DECtape holds up to 380,000 characters of information. DECtape data files may be dismounted for safekeeping, thereby insuring their privacy. Data files on DECtape are similar to files on disk except that they do NOT have filenames. Each reel of DECtape is treated as a discrete data file. When the tape is

mounted on a DECtape drive, records may be read and written directly onto the tape.

A DECtape data file may be used by only one user at a time. Once a DECtape unit is assigned, a single user has exclusive access to it until he releases it. Each DECtape drive has a WRITE LOCK switch which physically prevents any write operations to that unit. If the WRITE LOCK switch is set, programs may not write on the tape even if the unit is assigned.

DECtape data files may be used in a variety of ways. Programs which need large data files should use DECtape to avoid consuming large disk areas. Administrative files, such as student or employee records, are best stored on DECtape. Since they are removable and can be write-locked when mounted, their use can be tightly controlled. DECtapes are also useful for information retrieval. A data tape may be kept permanently mounted but write-locked. Individual users may run programs which assign and query that file, then release it for others to use.

4.4.1 DECtape File Records

Records for DECtape data files are specified the same way as for disk data files: with a RECORD statement. All rules for disk records apply to DECtape records. In fact, the same RECORD statement may be used for both a DECtape and disk file. (This is useful when transferring a tape file to a disk file for processing. Access to disk data files is considerably faster than to DECtape data files.)

It is possible to specify any record length for a DECtape data file, but a size of 43 variables is suggested, even more strongly than for disk data files. DECtapes are physically structured into blocks, each of which holds exactly 43 variables. If the record specified by the program is, for example, 44 variables, it requires two full blocks on the tape.

Records which are multiples of 43 variables are efficient in utilizing DECtape space but are not efficient in speed. Such records are written in consecutive DECtape blocks. The tape unit cannot read or write consecutive blocks without stopping the tape and rewinding it slightly (rocking). This tape rocking also occurs when single block records (43 variables or less) are read or written as consecutive DECtape records. (In this case, each DECtape file record corresponds to a physical tape block.)

The most efficient way to utilize DECtape is to make records 43 variables in length and write them onto every tenth record in the file (records 0, 10, 20, etc.). When the entire length of the tape has been traversed (the last block of the tape is number 1473), write next into records 1, 11, 21, etc. In this way, every record is eventually filled. Programs which will be used repeatedly should access the tape in this manner.

4.4.2 Opening a DECtape File

DECtape data files, like disk files, are completely separate from the programs which use them. Therefore, the program may specify which tape, or tapes, it will use. The OPEN statement is used for this purpose. Since DECtape files do not have names,[2] the OPEN statement specifies the DECtape unit number to be used. It is assumed that the proper tape reel has been mounted. If the file is to be updated, the unit should be write-enabled. If not, it should be write-locked. The OPEN statement is followed by the unit number to be used (0-7).

```
100 OPEN 2
100 OPEN 7
```

[2] It is important to note that BASIC data file DECtapes are not the same as the file-oriented DECtapes used by COPY. There is no directory on a BASIC DECtape file. Each tape is considered to be one file of data.

The unit number could be an expression. Making the unit number a variable is very useful since it is hard to predict which units will be available at the time the program is run. When the unit specification is a variable, the user may mount the file on any free unit, then INPUT the number into the program.

When the OPEN statement is executed, the indicated DECtape unit is automatically assigned to the user. It cannot subsequently be assigned to any other user. Thus, it is possible to try to open, hence assign, a unit which is already assigned. If, in the above examples, units 2 and 7 were already assigned to the current user or any other user, the program would be terminated and an error message printed.

An alternative form of the OPEN statement allows the program itself to handle this situation. OPEN statements may include an ELSE clause which specifies a line number. If the OPEN statement fails, BASIC automatically performs a GOTO to this line number. For example:

```
100 OPEN 2 ELSE 900
```

If unit 2 is available, it is assigned and BASIC goes on to execute the next statement. If unit 2 is not available, statement 900 is executed next. Statement 900 could print a message and perhaps ask for an alternate unit number.

4.4.3 Reading/Writing DECtape Files

DECtape data files are read and written using the same GET and PUT statements as are used for disk data files. The internal file number is a number between 0 and 7, or an expression. Unlike disk data files, DECtape data files are of a constant length equal to the capacity of the tape. The exact number of records per reel depends on the record size as follows:

Record Size	Tape Capacity
1-43 variables	1474 records
44-86 variables	737 records
87-129 variables	491 records

As indicated in the section on DECtape data records, a record size of 43 variables or less is recommended since it conforms to the physical blocking of the tapes themselves. It is also desirable to space the records along the tape so that the tape does not waste time rocking. The following subroutine could be used to write 1474 records on the tape in this fashion. It assumes that R is set to zero before it is called the first time and that the unit number is in U.

```
500 REM SUBROUTINE TO WRITE RECORDS ALONG TAPE
510 REM WRITES ONE RECORD EACH TIME CALLED
515 PUT U,10,R 'REMEMBER THIS INCREMENTS R
517 LET R=R+9 'SPACE OUT 10 BLOCKS
524 IF R<1474 THEN 550 'OK TO RETURN
530 IF R=1479 THEN 560 'TAPE IS FULL
540 LET R=R-1479
545 IF R>0 THEN 550
547 LET R=R+10
550 RETURN
560 STOP 'TAPE IS FULL
```

The following function may also be used to convert a logical record number (0 to 1469) to a physical record block spaced along the tape. This function does not use blocks 0-3. These blocks are, therefore, available for a header or label. Both the subroutine above and the function below assume a record length of 43 variables or less.

```
FNC(X) = (X-INT(X/147)*147)*10 + INT (X/147)+4
```

Once opened, any record on the tape may be read. The tape unit must, however, be write-enabled if it is to be written. Trying to PUT to a write-locked tape is an error.

4.4.4 Closing DECtape Files

Once all work on a DECtape data file has been completed it may be closed. Closing a file releases the tape unit and makes it available to other users. Thus, if the tape contains important information (and especially if it is write-enabled) the CLOSE should not be done until the tape reel has been removed. If no CLOSE statement is encountered in the program, the unit remains assigned after the program has finished. The DECtape unit remains assigned until a Monitor RELEASE command is executed or the user logs out. An example of a CLOSE statement follows:

```
1100 CLOSE 6
```

4.4.5 Using Dectape Data Files with OS/8 FORTRAN

Numeric DECtape data files written by BASIC may be read by OS/8 FORTRAN with the FORTRAN RTAPE and WTAPE subroutines, and vice-versa. (String and Hollerith variables use different character codes.) Thus, it is possible to use BASIC to prepare an input or update tape for a stand-alone FORTRAN program. This provides a convenient way to do big jobs in off-hours, without having to leave the time-sharing mode for very long.

4.5 LINE PRINTER OUTPUT

If a line printer is available, it may be used both to list BASIC programs and to serve as an output device for the programs themselves. The line printer may only be used by one user at a time. The statements associated with line printer output are LLIST and LPRINT.

LLIST is similar to the LIST command except that the program listing is output to the line printer rather than to the Teletype. The LLIST command assumes that no other user has the line printer

assigned and responds by typing WHAT? if the line printer is not available. After the listing is complete, the line printer is released and is available to any user.

BASIC programs may use the line printer as an output device during execution by means of the LPRINT statement. LPRINT is exactly like PRINT except that the information goes to the line printer rather than to the Teletype. All formatting conventions of the PRINT statement are available with LPRINT. In particular, CHR$(12) may be used to skip to the top of the next form (page).

The LPRINT statement also assumes that no other user has the line printer assigned. However, using this statement when the line printer is not available causes the program to terminate. Once LPRINT successfully assigns the line printer, it remains assigned until the program terminates.

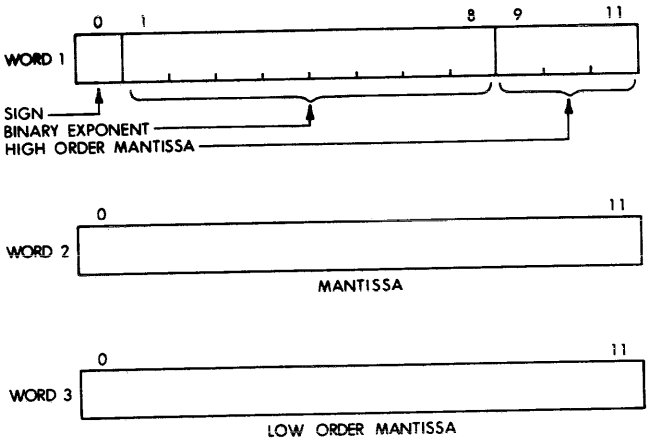
The OPEN and CLOSE statements may be used to assign and release the line printer. An OPEN statement with a device number of 11 assigns the line printer, or if it is not available and an ELSE clause is specified, transfers control to the line number specified in the ELSE clause. CLOSE 11 releases the line printer.

4.6 INTERNAL DATA CODES

Using the file I/O capabilities and the CHANGE statement, it is possible to examine data which was written on a DECtape or disk file by a program other than BASIC. There are two data formats: numeric and string.

4.6.1 Numeric Data

Each numeric value in BASIC is three PDP-8 words long. The storage format for numeric data is as follows:



A one in the sign bit means that the number is negative. The exponent is kept in excess 200 form where:

200_8 is 2^0
201_8 is 2^1
177_8 is 2^{-1}

The assumed decimal point is between bit 8 and 9 of word 1. Also, the number is always normalized, meaning that bit 9 is always 1 unless the number is zero. (Zero is represented by three zero words.) Note that this format is the same as the format used by OS/8 FORTRAN II; it is not the same as FORTRAN-D format.

4.6.2 String Data

Each string variable is three PDP-8 words long. Each word contains two 6-bit bytes or characters. If a string variable is filled by a GET from a source which was not written by a BASIC program, a BASIC program may examine the data in the variable by performing a CHANGE on that variable. The six bytes will be translated as if they were internal character codes for BASIC string characters. Table 4-1 shows how this translation interprets the 64 possible types. Note that after such a CHANGE, the 0th element of the array contains a count of the number of characters occurring before the first null.

TABLE 4-1

BASIC INTERNAL DATA CODES

6-Bit Byte Octal	Byte Decimal	ASCII Char.	6-Bit Byte Octal	Byte Decimal	ASCII Char.
00	0	NULL	40	32	?
01	1	SPACE	41	33	@
02	2	!	42	34	A
03	3	"	43	35	B
04	4	#	44	36	C
05	5	$	45	37	D
06	6	%	46	38	E
07	7	&	47	39	F
10	8	'	50	40	G
11	9	(	51	41	H
12	10	)	52	42	I
13	11	*	53	43	J
14	12	+	54	44	K
15	13	,	55	45	L
16	14	-	56	46	M
17	15	.	57	47	N
20	16	/	60	48	O
21	17	0	61	49	P
22	18	1	62	50	Q
23	19	2	63	51	R
24	20	3	64	52	S
25	21	4	65	53	T
26	22	5	66	54	U
27	23	6	67	55	V
30	24	7	70	56	W
31	25	8	71	57	X
32	26	9	72	58	Y
33	27	:	73	59	Z
34	28	;	74	60	[
35	29	<	75	61	\
36	30	=	76	62	]
37	31	>	77	63	↑

4.7 ERROR MESSAGES

Most programs, especially if they are at all complex do not execute correctly the first time they are tried. EduSystem 50 checks all BASIC statements when they are entered and before executing them.

If it cannot execute a statement, it informs the user by printing one of the error messages shown in Table 4-2, followed by the line number, if present, in which the error occurred.

In addition, the system checks for non-fatal execution errors and notifies the user that he performed a computational range error. When errors of this type occur, the messages shown in Table 4-3 appear, followed by the line number in which the error occurred.

TABLE 4-2

BASIC ERROR MESSAGES

Message	Explanation
ABORT	BASIC can not run for some reason. Perhaps the user's disk quota is exceeded.
ARRAY OR RECORD USED BEFORE DEFINITION	The RECORD statement must occur before any reference to it is made. A DIM statement must occur before an array is used. (RECORD and DIM are placed at the beginning of a program.)
BAD FILE FORMAT	The program specified in response to OLD PROGRAM NAME was not acceptable to BASIC. This is generally caused to load a non-BASIC (FORTRAN or PAL-D) program.
BAD FILE NAME	The file name used is not valid, e.g., it does not begin with a letter.
BAD SLEEP ARGUMENT	The argument of the SLEEP statement must have a number greater than or equal to 0, and less than or equal to 4094.
BAD VALUE IN CHANGE STATEMENT	While performing CHANGE A TO A$, one of the elements of the array A was found to contain an illegal value.
CAN'T CREATE FILE	An OPEN statement tried to create a file, but there is: (a) no disk space available, (b) no file name specified, or (c) a null string has been given as the file name.

Table 4-2 (Cont.)

Message	Explanation
CAN'T DELETE FILE	UNSAVE cannot delete a file. This is usually due to the fact that another user has the file open, or the file is protected with a code $\geq$20.
CAN'T DELETE: name	An attempt has been made to UNSAVE a protected file.
CAN'T FIND LINE	An attempt has been made to edit a non-existent line.
CAN'T FIND "name" IN SYSTEM LIBRARY	The requested old file cannot be found.
CHAIN TO BAD FILE	The file specified by the CHAIN has an invalid format: it is not a BASIC format file. The "PROGRAM IS..." message will follow this error message. The program name will be the name of the bad file.
DEF STATEMENT MISSING	A function needing a DEF statement exists in the program.
DEVICE BUSY	The user tried to OPEN DECtapes 0-7, line printer, or paper tape punch, but the device was unavailable, and there was no ELSE clause in the OPEN statement.
DIMENSION TOO LARGE	Too large an array to fit in the available core.
DISK FULL	There is no more storage space on the system disk.
DUPLICATE FILE NAME	An attempt has been made to SAVE a program, but one with that name already exists.
EXECUTE ONLY	An attempt has been made to list, save, or alter a BASIC complied program. The program can only be run.
FOR WITHOUT NEXT	There is an unmatched FOR statement in the program.
GET BEYOND END OF FILE	Disk data file is too small to have a record with the number specified in the GET statement at line n.

Table 4-2 (Cont.)

Message	Explanation
GET/PUT ERROR	A hardware error occurred in GET or PUT. (This is usually due to a DECtape unit being write-locked.)
GOSUB - RETURN ERROR	Subroutines are too deeply nested or a RETURN statement exists outside a subroutine.
ILLEGAL CHARACTER	The user attempted to use an illegal character in the statement being processed.
ILLEGAL CONSTANT	The format of a constant in the statement being processed is not valid.
ILLEGAL FORMAT	The structure of the statement does not agree with BASIC syntax.
ILLEGAL FOR NESTING	FOR-NEXT loops are too deeply nested or NEXT appears before FOR.
ILLEGAL INSTRUCTION	A statement was used which is not one of the legal BASIC statements.
ILLEGAL LINE NUMBER	The format of the line number being used in a GOTO or IF statement is not acceptable. A line number has been typed which is not between 1 and 2046, inclusive.
ILLEGAL OPERATION	The expression being processed does not agree with the BASIC rules (this is probably due to unmatched parentheses).
ILLEGAL SYNTAX	The expression in a statement does not agree with the BASIC syntax.
ILLEGAL VARIABLE	An illegal variable was used in an array.
IMPROPER ACCOUNT # ABORT ↑BS	A user logged in under account numbers 1 (system account) or 2 (system library) and tried to run BASIC. This is prohibited.
IMPROPER DIM OR RECORD STATEMENT	Syntax error in DIM or RECORD statement, or an array name that was previously dimensioned is reused.

Table 4-2. (Cont.)

Message	Explanation
INVALID DEVICE NO.	The device number in the file I/O statement is not between 0 and 11 inclusive, (or X and 11 inclusive where X is a number set by the system manager).
INVALID RECORD NO.	The record number must be a number which is greater than or equal to 0 and less than or equal to 4095. For DECtape I/O the maximum record number is limited further by the DECtape size.
LINE TOO LONG	Too much has been typed.
MISUSE OF CHR$	The CHR$ function was used in an invalid manner. CHR$, like TAB, can appear only in PRINT statements.
MISUSED TAB	The TAB function was used in an invalid manner. TAB can appear only in PRINT statements.
MORE?	Not enough values have been entered in response to an INPUT command. The rest of the values may be entered.
NEXT WITHOUT FOR	The NEXT statement indicated has no preceding FOR statement.
NO END STATEMENT	All programs must have an END statement.
ON INDEX OUT OF RANGE	The value of the index is less than one, or greater than the number of statement numbers.
OUT OF DATA	An attempt was made to READ more data than was supplied by the user.
PROGRAM IS "progname"	This message may immediately follow an error message, to identify the current program in a series of CHAINed programs. If there is no CHAIN, this message will not occur.
PROGRAM NOT FOUND	The file which the user tried to access with a CHAIN statement does not exist in his disk area. The PROGRAM IS message will also occur.
PROGRAM TOO LARGE	The program is too large to be executed. Make it smaller.

Table 4-2. (Cont.)

Message	Explanation
STACK OVERFLOW	The user programmed a situation in which the expression is too complicated to be executed.
SUBSCRIPT ERROR	A negative subscript was used for an array.
SYSTEM I-O ERROR	BASIC was unable to perform the desired disk I/O.
TIME LIMIT EXCEEDED	The number of statements executed by a job has exceeded the maximum established by the system manager. Generally, some error was made and the program is caught in a loop.
TOO MUCH INPUT, EXCESS IGNORED	Too many values have been entered in response to an INPUT command.
UNDEFINED LINE NUMBER	The line number appearing in a GOTO or an IF-THEN statement does not appear in the program.
UNOPEN DISK UNIT	The user tried to do a GET, PUT, or UNSAVE to device 8 or 9, without a file being previously opened on the device
WHAT?	The editor cannot understand the command given.

TABLE 4-3

NON-FATAL EXECUTION ERRORS

Message	Explanation
IC	An illegal character or constant has been typed in response to an INPUT statement. The input is requested again.
LN	An attempt was made to compute the logarithm of zero or a negative number. Zero is used for the result.
OV	Overflow - the result of a calculation was too large for the computer to handle. The largest possible number is used for the result.
PW	An attempt was made to raise a negative number to a fractional power. The absolute value of that number raised to the fractional power is used.
SQ	An attempt was made to compute the square root of a negative number. The square root of the absolute value is used for the result.
UN	Underflow - the result of a calculation was too small for the computer to handle. Zero is used for the result.
/0	Zero divide - an attempt was made to divide by zero. The largest possible number is used for the result.

TABLE 4-4

EDUSYSTEM 50 BASIC LANGUAGE SUMMARY

Statement	Format	Explanation
Input/Output Statements		
CLOSE	CLOSE f	Close file f.
DATA	DATA $n_1,n_2,...n_n$	Numbers n_1 through n_n = variables in READ statement.
GET	GET f,l,r	Read record r from file f into the variables in line l.
INPUT	INPUT $v_{1,2,...v_n}$	Get v_1 through v_n input from the terminal.
LINPUT	LINPUT $v\$_1,v\$_2,...v\$_n$	Get long character string input from terminal (up to 50 characters).
LPRINT	LPRINT $e_1,e_2,...e_n$	Print values of specified text or expressions on line printer.
OPEN	OPEN f,n$	Open a file named n$ as file number f.
OPEN-ELSE	OPEN f ELSE n	Open a file, go to line n if unavailable.
PRINT	PRINT $e_1,e_2,...e_n$	Print values of specified text, variables, or expressions.
PUT	PUT f,l,r	Write record r, formatted as in line l, into file f.
READ	READ $v_1,v_2,...v_n$	Read variables v_1 through v_n from DATA list.
RESTORE	RESTORE	Reset DATA pointer to beginning value.
RESTORE*	RESTORE*	Reset DATA pointer for numeric data only.
RESTORE$	RESTORE$	Reset DATA pointer for character string data only.
UNSAVE	UNSAVE f	Delete file f from disk storage.

Table 4-4. (Cont.)

Statement	Format	Explanation
Transfer of Controls		
GOTO	GOTO 1	Transfer control to line number 1.
IF-GOTO	IF e_1 r e_2 GOTO 1	If relationship r between e_1 and e_2 is true, transfer control to line number 1.
IF-THEN	IF e_1 r e_2 THEN 1	Same as IF-GOTO.
ON-GOTO	ON e GOTO 1_1,1_2,...1_3	Computed GOTO.
Loops and Subscripts		
DIM	DIM v(d_1), v(d_1,d_2)	Dimension subscripted variables.
FOR-TO-STEP	FOR v=e_1 TO e_2 STEP e_3	Set up program loop. Define v values beginning at e_1 to e_2, incremented by e_3.
NEXT	NEXT v	Terminate program loop. (Increment value of v until v>e_2.)
Subroutines		
GOSUB	GOSUB 1	Enter subroutine at line 1.
RETURN	RETURN	Return from subroutine to statement following GOSUB statement.
STOP	STOP	Transfer control to END statement.
Others		
CHAIN	CHAIN n$	Link to next program.
CHANGE	CHANGE v_1 TO v_2	Change character string to array of character codes or vice versa.
DEF	DEF FNA(x)=f(x)	Define a function.
END	END	End of program.

Table 4-4. (Cont.)

Statement	Format	Explanation
LET	LET v=f	Assign value of formula f to variable v.
RANDOMIZE	RANDOMIZE	Randomize random number routine.
REMARK	REM text	Insert a remark or comment.
SLEEP	SLEEP n	Cause program to pause for n seconds.
Edit/Control Commands		
BYE	BYE	Leave BASIC Monitor.
CATALOG	CAT	List names of programs in storage area.
COMPILE	COM name	Compile program in core and save it on disk.
CTRL/C	↑C	Stop program execution; return to edit phase.
DELETE	DEL n	Delete line n.
	n	Delete line n.
	DEL n,m	Delete lines n through m, inclusive.
EDIT	EDI n	Search line n for character c.
	(c)	
KEY	KEY	Return to keyboard mode after TAPE.
LIST	LIST	List entire program in core.
	LIST n	List line n only.
	LIST n,m	List lines n through m, inclusive.
LLIST	LLIST	List program on line printer.
NEW	NEW	Clear core, request program name.
OLD	OLD	Clear core, bring program to core from storage area.

Table 4-4. (Cont.)

Statement	Format	Explanation
REPLACE	REP REP name	Replace old file on disk with version in core. If name is not specified, old name is retained.
RUN	RUN	Compile and run program in core.
SAVE	SAVE name	Store program named on storage device.
SCRATCH	SCR	Erase current program from core.
TAPE	TAP	Read paper tape; suppress printing on Teletype.
UNSAVE	UNSAVE name	Delete program named from storage device.
Functions		
ABS	ABS(x)	Absolute value of x.
ATN	ATN(x)	Arctangent of x (result in radians).
COS	COS(x)	Cosine of x (x in radians).
EXP	EXP(x)	e^x (e is approximately 2.7182818).
INT	INT(x)	Greatest integer of x.
LOG	LOG(x)	Natural logarithm of x.
RND	RND(x)	Random number.
SGN	SGN(x)	Sign of x (+1 if positive, -1 if negative, 0 if zero).
SIN	SIN(x)	Sine of x (x in radians).
SQR	SQR(x)	Square root of x.
TAN	TAN(x)	Tangent of x (x in radians).
TAB	TAB(x)	Controls printing position on terminal or line printer.
FIX	FIX(x)	Truncates decimal portion of x.
CHR$	CHR$(x)	Converts character code to character. Used only with the PRINT or LPRINT statements.

CHAPTER 5

FOCAL

FOCAL (FOrmula CALculator) is an on-line, interactive, service program for the PDP-8 family of computers, designed to help scientists, engineers, and students solve numerical problems. The language consists of short imperative English statements which are easy to learn. FOCAL is used for simulating mathematical models, for curve plotting, for handling sets of simultaneous equations, and for many other kinds of problems.

To call FOCAL, type:

```
.R FOCAL
```

FOCAL enters its initial dialogue, and asks if its extended functions are to be retained. The extended functions are exponential, sine, cosine, arctangent, and logarithm. If the FOCAL program to be run uses any of these functions, the user responds YES. If not, the user responds NO to free more space for the user program. Without the extended functions, there is room for approximately 1100 characters of program. If the extended functions are retained, there is room for approximately 735 characters.

5.1 USING FOCAL COMMANDS

Whenever FOCAL prints an asterisk, it is in command mode, and the user may type any of the FOCAL commands in response to the asterisk. FOCAL commands may be direct or indirect. A direct command is typed directly after the asterisk and is executed immediately. The format for direct commands is:

*COMMAND

An indirect command is always identified by a line number. Indirect commands are not executed until program control passes to the line number associated with the command. The format for indirect commands is:

*GG.ss COMMAND

When the user is typing indirect commands, he may use any line number in the range 1.01 to 31.99, except those ending in .00. Numbers such as 1.00 or 31.00 are illegal as line numbers; they are used to identify an entire group of line numbers. Line numbers are typed in the format:

GG.ss

where GG is the group number and ss is the step number. It is not necessary to type two digits after the decimal; e.g., 2.1 is equivalent to 2.10.

All FOCAL commands must be followed immediately with a space. All FOCAL command lines must be terminated with the RETURN key.

5.2 FOCAL OVERVIEW

FOCAL consists of 12 commands which are all the beginner needs to write programs. FOCAL commands may be typed in their entirety or abbreviated. The FOCAL commands are:

Command	Explanation
ASK	Used to assign values to variables from the keyboard.
COMMENT or CONTINUE	Used for comments or non-executable program steps.
DO	Used to cause a specific line or group of lines to be executed.
ERASE or ERASE ALL	Used to erase part of a program or an entire program.
FOR	Used to increment a number and execute a user-specified command for each value of the number incremented.
GO or GOTO	Used to direct program control to the lowest line number, or to some specific line number.
IF	Used to direct program control conditionally after a comparison.
MODIFY	Used to edit words or characters on a program line.
QUIT	Used to halt program execution and return control to user.
RETURN	Used to terminate DO routines.
SET	Used to define variables and evaluate expressions.
TYPE	Used to print text, results of calculations, and values of variables.

These commands are explained in detail with actual computer output in this section. For the convenience of the user, a detailed FOCAL command summary is included in Table 5-1.

5.3 NUMBERS

A FOCAL number may be any decimal number between -10^{615} and 10^{615}. Numbers may be written signed (+ or -) or unsigned, either with a decimal point and a fractional part or in exponential format (see Data Formats) with a mantissa and exponent. In FOCAL, all numbers are internally represented in exponential format, retaining

up to six significant digits. If more than six digits are specified, the number will be rounded to six digits. The following numbers are identical in FOCAL:

```
60
60.00
6E1
600E-1
60.00003
```

5.4 VARIABLE NAMES

FOCAL variable names may consist of either one or two characters. The first character must always be alphabetic; however, it cannot be an F because FOCAL reserves that character for function names (see FOCAL Functions). The second character may be either alphabetic or numeric. The user may write variable names consisting of more than two characters, but FOCAL uses only the first two characters to identify the variable.[1] Therefore, the first two characters must be unique.

```
*SET A=56789
*SET B=123456
*SET C1=15
*SET C2=30
*SET DEPTH=10
*SET DISTANCE=C1+C2
```

Variables may also be subscripted. For a discussion of what subscripted variables are and how they are used, see Subscripted Variables.

---

[1] A variable is represented internally as a binary fraction with an exponent. See Data Formats.

5.5 ARITHMETIC OPERATIONS

To print the results of arithmetic calculations, the user types the FOCAL command TYPE followed by a space and the data to be calculated. Then he presses the RETURN key, and FOCAL prints the answer. For example:

```
*TYPE 6+10-3-1
=   12.0000*
```

The above example shows two of the arithmetic operations (+ and -) performed by FOCAL. Arithmetic operations are performed from left to right except when the operation to the right has priority or when enclosures are used. (See Enclosures.)

```
*TYPE 6+5; TYPE 5+2-3; TYPE 10-6
=   11.0000=    4.0000=    4.0000*
```

NOTE

Several commands may be typed on the same line if they are separated by semi-colons (;). This is true for all FOCAL commands except the LIBRARY commands.

Unless indicated otherwise, FOCAL mathematical computations retain an accuracy of six significant digits.

5.5.1 Priority of Arithmetic Operations

The FOCAL arithmetic operations priorities are:

First priority - exponentiation (↑)[2]

Second priority - multiplication (*)

[2]When exponentiation is performed by FOCAL, the power to which a number is raised must be a positive integer. If a calculated exponent exceeds the limits of size, no error message is given. The result will go to zero.

Third priority - division (/)
Last priority - addition (+), subtraction (-)

When FOCAL evaluates an expression which includes several arithmetic operations, the above order of priority is followed. Therefore, FOCAL evaluates

```
*TYPE 25+5*2+5
```

to have a value of

```
=   40.0000*
```

because multiplication (*) has a higher priority than addition (+).

5.5.2 Enclosures

The order of executing arithmetic operations is also influenced by enclosures. Three kinds of enclosures may be used with FOCAL: parentheses ( ), square brackets [ ][3], and angle brackets< >. FOCAL treats them all the same. For example, the result of the expression:

```
*TYPE (A+B)*<C+D>*[E+F]
```

is the same as the result of:

```
*TYPE (A+B)*(C+D)*(E+F)
```

If the expression contains enclosures within enclosures (called nesting), FOCAL executes the contents of the innermost enclosures first and works outward.

```
*TYPE (5*<2+3>-(5)↑2
=  400.0000*
```

[3] [and] are typed by SHIFT/K and SHIFT/M, respectively.

5.6 INPUT/OUTPUT COMMANDS

5.6.1 TYPE Command

The TYPE command is used to print results of calculations, values of variables, text or character strings, and variable tables. TYPE may also be used to print combinations of text and variables.

Example 1 - Result of a Calculation:

```
*TYPE 1+1
=     2.0000*
```

Example 2 - Value of a Variable or Variables:

```
*SET N=5+5; SET M=30
*TYPE N,M
=    10.0000=    30.0000*
```

Example 3 - Text:

```
*TYPE "THIS IS A LINE OF TEXT", !4
THIS IS A LINE OF TEXT
*
```

Example 4 - Variable Tables:

```
*TYPE $
N@(00)=    10.0000
M@(00)=    30.0000
*
```

The user may command FOCAL to print all of the user-defined variables (variable table) by using the TYPE command and a dollar sign ($).

---

[4] The exclamation mark (!) causes a carriage return and line feed.

If a variable consists of only one letter, an at sign (@) is inserted as a second character in the variable table printout, as shown in the example above.

```
*SET N=25
*TYPE "N IS",N
N IS=   25.0000*
```

NOTE

> Any variable, constant, or expression in a TYPE or ASK command must be followed by a comma, semicolon, or carriage return.

5.6.2 ASK Command

The ASK command is normally used in indirect commands to enable the user to input numerical data during the execution of his program. The ASK command is similar to the TYPE command in form, but only single variable names, not expressions, are used; and the user types the values in response to a colon (:) printed by the ASK command.

```
*11.99 ASK X,Y,Z
```

When FOCAL encounters line 11.99 in the above example, it prints a colon and waits for the user to type a value (in any format) and a terminator[5] for the first variable. This process continues until all the variables in the ASK command have been given values.

The value is assigned to the variable when the user types a terminator; so any time before the terminator is typed, the value can be changed. If the user types back arrow (← or SHIFT/O) immediately after the value and before he types a terminator, he can then type the correct value and a terminator.

---

[5] Terminators are SPACE, comma, ALT MODE, and RETURN keys. If the user types the RUBOUT key, it is ignored.

```
*ASK X,Y,Z
:5                              User typed 5 and RETURN key.
:6                              User typed 6 and RETURN key.
:8←7                            User typed 8,← , 7 and RETURN key.
*TYPE X,!,Y,!,Z
=     5.0000
=     6.0000
=     7.0000*
```

The ALT MODE key is a special non-spacing terminator which enables the user to have a previously assigned value unchanged.

```
*ASK X,Y,Z
:3
::12                            User typed ALT MODE because he did
*TYPE X,!,Y,!,Z                 not want to change the value of Y.
=     3.0000
=     6.0000
=    12.0000*
```

5.6.2.1 Text Output with ASK -- The ASK command, just as the TYPE command, may be used to print text. Carriage return and line spacing are controlled the same as with the TYPE command (see Data Formats).

```
*ASK "WHAT IS YOUR AGE?" AGE
WHAT IS YOUR AGE?:19
*
```

The word following the text in the command line (AGE, in this example) is the variable.

5.7 COMPUTATIONAL COMMAND (SET)

The SET command enables the user to assign a numerical value to a variable and store both the value and the variable. Then, when he uses the variable in an expression[6], FOCAL automatically substitutes the numerical value that the user previously specified:

[6] An expression is a combination of arithmetic operations or functions which may be reduced to a single number by FOCAL.

```
*SET E=2.71828
*SET PI=3.14159
*TYPE "PI TIMES E",PI*E
PI TIMES E=    8.5397*
```

The value of a variable may be changed at any time by another SET command.

```
*SET A1=3+2
*SET A1=A1+1
*TYPE A1
=     6.0000*
```

5.8 CONTROL COMMANDS

5.8.1 GO or GOTO Command

The GO command causes FOCAL to go the lowest numbered line in the program and begin executing the indirect commands.

```
*1.1 SET A=1
*1.3 SET B=2
*1.5 TYPE A,B
*GO
=     1.0000=     2.0000*
```

In the above example the GO command caused execution to begin at line 1.1.

The GOTO command causes FOCAL to go to a specific line in the program and begin executing the indirect commands in ascending line number order.

```
*ERASE
*1.1 SET A=1
*1.3 SET B=2
*1.5 TYPE A,B
*GOTO 1.3
=     0.0000=     2.0000*
```

In the preceding example, A and B are equal to zero at the start of the program. The ERASE and ERASE ALL commands are used to ensure that all variables are equal to zero until they are assigned a specific value. Since the GOTO command causes program execution to begin at line 1.3, line 1.1 is never executed and A is not set to 1.

5.8.2 IF Command

The IF command is a conditional command used to transfer program control after a comparison. The normal IF command format is:

IF space (expression) line1, line2, line3

The expression or variable is evaluated, and program control is transferred to the first line number if the value of the expression is less than zero, to the second line number if the value is zero, or to the third line number if the value is greater than zero.

The program below transfers control to line number 2.1, 2.3, or 2.5, according to the value of the expression in the IF command.

```
*2.1 TYPE "LESS THAN ZERO";QUIT
*2.3 TYPE "EQUAL TO ZERO";QUIT
*2.5 TYPE "GREATER THAN ZERO"; QUIT
*IF (25-25)2.1,2.3,2.5
EQUAL TO ZERO*
```

5.8.2.1 IF with Less Than Three Line Numbers -- The IF command format can be altered to transfer program control to one or two lines. For example, if a semicolon or a carriage return is immediately after the first line number, control goes to the first line number if the value of the expression is less than zero. If the value is not less than zero, control goes to the next sequential command. For example:

```
*2.20 IF (X) 1.8;TYPE "Q"
```

When line 2.20 is executed, program control goes to line 1.8 if X is less than zero. If X is not less than zero, Q is typed.

If a semicolon or a carriage return follows the second line number, control goes to the first or second line number, depending upon whether the value of the expression is less than zero or equal to zero. If the value is greater than zero, control goes to the next sequential command. For example:

```
*3.19 IF (B)1.8,1.9
*3.20 TYPE B
```

If B is less than zero, control goes to line 1.8; if B equals zero, control goes to 1.9; and if it is greater than zero, control goes to the next sequential command, in this case line 3.20, and the value of B is printed.

5.8.2.2 Arithmetic Comparison with IF Command -- The IF command can be used with all the arithmetic operations of FOCAL.

Example 1 - Addition:

```
 *1.10 IF (A+B)2.1,2.5,2.10
```

Example 2 - Subtraction:

```
*5.16 IF (A-B)1.6;TYPE "X"
```

Example 3 - Division:

```
*3.10 IF (M/N)5.5,5.6
*3.15 TYPE "GREATER THAN ZERO"
```

Example 4 - Multiplication:

```
*7.20 IF (P*I)8.1
*7.25 TYPE P*I
```

Example 5 - Exponentiation:

```
*4.15 IF (X↑N)6.1,6.2,6.3
```

5.8.3 DO Command

The DO command is used to make subroutines of single lines or groups of lines. Control is returned to the line following the DO command after the subroutine is executed.

```
*1.1 SET A=1;SET B=2
*1.2 TYPE "STARTING"
*1.3 DO 3.2
*2.1 TYPE "FINISHED"
*3.1 SET A=3;SET B=4
*3.2 TYPE A+B
*GO
STARTING=     3.0000FINISHED=     7.0000*
```

If the user types a command such as DO 3, the DO command treats the group of program lines beginning with 3 as a subroutine. Control proceeds in ascending order through the group numbers until the end of the group is reached, or until a RETURN command is executed.

5.8.3.1 Nested DO -- DO commands may be nested as shown in the following example:

```
*1.1 TYPE "BEGIN",!
*1.2 DO 2
*1.3 TYPE "END",!;QUIT
*
*2.1 DO 5.1; TYPE A,!
*2.2 DO 5.2; TYPE A,1\!
*2.3 DO 7.5; TYPE A,!
*
*5.1 SET A=1
*5.2 SET A=2
*
*7.5 SET A=3
*GO
BEGIN
=     1.0000
=     2.0000
=     3.0000
END
*
```

The number of nested DO commands is limited only by the amount of core memory available after storage is allocated for program and variables.

5.8.4 RETURN Command

The RETURN command is used to exit from a DO subroutine. When a RETURN command is encountered during execution of a DO subroutine, the program exists from its subroutine status and returns to the command following the DO command that initiated the subroutine status.

5.8.5 QUIT Command

When the QUIT command is executed, FOCAL prints an asterisk and returns to command mode.

5.8.6 FOR Command

The FOR command is used to set up program loops and iterative procedures. The general command format is:

```
*FOR A = B,C,D; commands
*FOR A = B,D; commands
```

The variable A is initialized to the value B, then the command or commands following the semicolon are executed. When the commands have been executed, the value of A is incremented by C and compared to the value of D. If A is less than or equal to D, the command after the semicolon is executed again. This process is repeated until A is greater than D, at which time FOCAL goes to the next sequential line. The command or commands will always be executed at least once.

A must be a single variable. B, C, and D may be expressions, variables, or numbers. If the value C is omitted, it is assumed that the increment is one. If C and D are omitted, the FOR command is handled like a SET command and no iteration is performed. A FOR command may be used with a DO command and nested:

5.8.6.1 FOR with a DO

```
*ERASE ALL
*1.1 FOR X=1,1,5; DO 2
*1.2 QUIT
*
*2.1 TYPE !,"   X",X
*2.2 SET A=X+100
*2.3 TYPE !,"   A",A
*GO

   X=    1.0000
   A=  101.0000
   X=    2.0000
   A=  102.0000
   X=    3.0000
   A=  103.0000
   X=    4.0000
   A=  104.0000
   X=    5.0000
   A=  105.0000*
```

5.8.6.2 Nested FOR and DO

```
*1.1 FOR Z=1,3; TYPE "  A  B  C  "
*1.2 TYPE !
*1.5 FOR A=1,3; DO 3
*1.7 QUIT
*
*3.1 FOR B=1,3; DO 4
*
*4.1 FOR C=1,3; TYPE %1,A,B,C,"  "
*4.2 TYPE !
*GO
  A  B  C    A  B  C    A  B  C
= 1= 1= 1  = 1= 1= 2  = 1= 1= 3
= 1= 2= 1  = 1= 2= 2  = 1= 2= 3
= 1= 3= 1  = 1= 3= 2  = 1= 3= 3
= 2= 1= 1  = 2= 1= 2  = 2= 1= 3
= 2= 2= 1  = 2= 2= 2  = 2= 2= 3
= 2= 3= 1  = 2= 3= 2  = 2= 3= 3
= 3= 1= 1  = 3= 1= 2  = 3= 1= 3
= 3= 2= 1  = 3= 2= 2  = 3= 2= 3
= 3= 3= 1  = 3= 3= 2  = 3= 3= 3
*
```

Another way of handling the same program is:

```
*1.1 FOR Z=1,3;TYPE "  A  B  C  "
*1.2 FOR A=1,3;FOR B=1,3;TYPE !;FOR C=1,3;TYPE %1,A,B,C,"  "
*GO
```

5.8.6.3 Subscripted Variables -- Variables may be further identified by subscripts which are enclosed in parentheses immediately following the variables. For example:

```
*SET AB(0)=5
*SET AB(1)=10
*SET AB(2)=15
*SET AB(3)=20
*SET AB(4)=25
*SET AB(5)=30
*FOR X=0,5; TYPE AB(X),!
= 0.500000E+01
= 0.100000E+02
= 0.150000E+02
= 0.200000E+02
= 0.250000E+02
= 0.300000E+02
```

In the above example, subscripts are used to set up an array called AB. Any element in the array can be represented by a subscript in the range 0 to 5. The first element in an array always has a subscript of 0. A subscript may be a number, another variable, or an expression. If it is a number, it must be in the range $\pm$2047. In order to be properly represented by the TYPE $ command, subscript numbers must be positive integers in the range from 0 to 99. The TYPE $ command will print subscripts greater than 99 as two random characters, although their contents will be correct as assigned by the program.

5.8.7 COMMENT or CONTINUE Command

The COMMENT or CONTINUE command (abbreviated as C) causes the program line to be ignored by FOCAL. The user may use the C command to insert comments into the program, or he may use the C command line as a non-executable program step. In either case, program lines beginning with C are skipped when the program is executed. However, comments are printed in response to a WRITE command.

```
*ERASE ALL
*1.1 C INITIALIZE VARIABLES
*1.2 SET A=5
*1.3 SET B=6
*1.4 SET C=7
*
*2.1 C PERFORM CALCULATION
*2.2 TYPE A+B+C
```

5.9 EDIT COMMANDS

5.9.1 WRITE or WRITE ALL Command

The WRITE or WRITE ALL command causes FOCAL to print a program line, a group of lines, or an entire indirect program on the terminal.

```
*WRITE 2.1          Print a single line.
*WRITE 2            Print a group of lines.
*WRITE              Print an entire program.
```

Once the program is completed, the user may want to save it by putting it on paper tape. The procedure for saving a FOCAL program on-line is as follows:

1. Make sure FOCAL is in command mode.
2. Type WRITE.
3. Set low-speed punch to ON position.
4. Type RETURN key.

FOCAL will punch the entire program onto the paper tape and simultaneously print it on the terminal.

Paper tapes may be read in from the Terminal Reader by following these instructions.

1) Make sure FOCAL is in command mode.

2) Place tape in Reader (Reader off).

3) Type CTRL/R.

4) Turn Reader on.

5) Turn Reader off when the tape reaches the Trailer.

6) Type CTRL/T.

Paper tapes may also be punched in the following manner.

1) Turn Terminal offline to Local.

2) Turn punch on.

3) Hold the REPEAT, CTRL, and SHIFT keys down. Now press the P and hold all four keys down until four or five inches of Leader are punched. Release the repeat key first.

4) Turn off the punch and return the Terminal to the line position.

5) Type CTRL/R, and turn the punch on.

6) Type W A, followed by a carriage return (this will not echo on the terminal).

7) When the tape is finished punching, turn the punch off, and type CTRL/T.

8) Repeat steps 1, 2, 3, and 4 for making Trailer.

5.9.2 ERASE and ERASE ALL Commands

The ERASE command deletes symbolic names, lines, or groups of lines.

```
*ERASE          Delete all names defined in symbol table.
*ERASE 2.2      Delete line 2.2.
*ERASE 2        Delete all lines in group 2.
```

The user can check to see if the line(s) has been deleted by typing the WRITE command after the ERASE command. This is a useful procedure for checking commands and also for obtaining a clean print-out of the current program.

The ERASE ALL command deletes the entire indirect program. It is good programming practice to type ERASE ALL before starting to type a new program. The ERASE ALL command is generally used only with direct commands because it returns control to command mode upon completion.

5.9.3 MODIFY Command

The MODIFY command is used to change characters within a line without changing the entire line. The format for MODIFY is:

MODIFY line number RETURN key Search character

The search character is not printed. After the user has typed the line number, PETURN key, and search character, FOCAL prints the contents of the specified line until it encounters the search character. When the search character is read and printing stops, the user has any one or more of the following options:

1. Type new characters in addition to those already printed.
2. Type a form feed (CTRL/L). This causes the search to proceed to the next occurrence, if any, of the search character.
3. Type CTRL/BELL. The user can then change the search character he specified in the MODIFY command.
4. Type the RUBOUT key. This causes FOCAL to delete a character, starting with the last character printed and deleting one character to the left each time RUBOUT is typed.
5. Type the back arrow (←) key. This causes FOCAL to delete everything between the back arrow and the line number.

6. Type the RETURN key. This causes FOCAL to terminate the line at that point, deleting everything to the right.

7. Type the LINE FEED key. This is normally done after the user has exercised one or more of the above options. After the user has modified the line, he may type LINE FEED and cause the remainder of the line from the search character to the end to be printed and saved.

```
*7.Ø1 JACK AND BILL W$NT UP THE HILL
*MODIFY 7.Ø1
JACK AND B\JILL W$\E
                    NT UP THE HILL
```

In the above example, B was typed as the search character for line 7.01. (Note that the search character did not print.) FOCAL stopped printing when it encountered the search character (B), and the user typed the RUBOUT key (\) to delete the B. Then he typed the correct letter J. Next he typed CTRL/BELL and the $ key to change the search character. FOCAL continued to print the line until the new search character was encountered. The user typed RUBOUT to delete the $ and then typed the correct character (E). He then typed the LINE FEED key and the remainder of the line was printed.

CAUTION

When any text editing is done, the values in the user's symbol table are reset to zero.

If the user defines his symbols in direct statements and then uses a MODIFY command, the values of his symbols are erased and must be redefined. However, if the user defines his symbols by means of indirect statements prior to using a MODIFY command, the values will not be erased because these symbols are not entered in the symbol table until the statements defining them are executed. Notice in the example below that the values of A and B were set using direct statements. However, typing the lines 1.1 through 1.3 reset these values to zero.

```
*ERASE ALL
*SET A=1
*SET B=2
*1.1 SET C=3
*1.2 SET D=4
*1.3 TYPE A+B+C+D; TYPE !; TYPE $
*GO
= 0.700000E+01
C@(00)= 0.300000E+01
D@(00)= 0.400000E+01
A@(00)= 0.000000E+00
B@(00)= 0.000000E+00
*
```

5.10 LIBRARY COMMANDS

In addition to the basic FOCAL commands, there are two special commands to perform the library functions: storing and retrieving data from the system disk. All commands following the LIBRARY command on the same line are ignored. Names of files may be from 1 to 6 characters long. Only alphanumeric characters, letters, and numbers should be used in file names. The library commands, like other FOCAL commands, may be abbreviated.

5.10.1 LIBRARY SAVE Command

This command copies the current program into the user's area on disk and gives it the name specified. For example, the command:

```
*L S TRUE
```

will cause the current program to be stored on disk under the name TRUE. The program will also remain in the user's area.

5.10.2 LIBRARY CALL Command

This command copies the named program from disk into the user's area. For example, the commands:

```
*L C TRUE
*W
```

will cause the program TRUE to be recalled from the disk into the user's area and listed on the terminal. Any program currently in the user's area will be erased. The program TRUE can then be executed with a GO command.

If the LIBRARY CALL command is given a line number and stored, it must have at least one numbered command following it. In this case, the LIBRARY CALL command will cause the named program to be called into the user's area and executed as if GO had been typed. For example:

```
*3.10 L C TRUE
*3.11 C
```

If the above commands are included in a program, they will cause TRUE to be brought in and executed at that point. If there are lines beyond 3.11 remaining to be executed, they will be deleted. This procedure allows the user to chain FOCAL programs as in BASIC.

5.10.3 Error Messages with Library Commands

When the LIBRARY commands are used, five errors are possible. These are also listed in the error code summary in Table 5-3.

Message	Explanation
?30.71	The command appeared to be a LIBRARY command but was not, for example: LIBRARY OPEN No action is taken; the command should be retyped.
?30.<0	Either an unacceptable file name was specified or no name was specified where one was required.
?31.42	The file name specified does not match any name currently in the user's disk directory. This error will only occur with the LIBRARY CALL command.
?31.43	There is already a program with the specified name in the directory. This error will only occur with the LIBRARY SAVE COMMAND.
?31.44	This user's disk directory is full. The current program cannot be saved until others have been deleted. This error will only occur with the LIBRARY SAVE command.

5.11 ESTIMATING PROGRAM LENGTH

FOCAL permits approximately 1100 (decimal) locations to be used for the user program and variables without the extended math functions, and approximately 735 locations with the extended functions (sine, cosine, log, exponential, etc.). Since FOCAL requires five locations for each variable stored in the variable table and one location for each two characters of stored program, the approximate length of a program may be determined by the formula:

$$\text{Length of program} = 5S + \frac{C}{2} + 2L$$

where

S = number of variables
C = number of characters in program
L = number of lines

If the total program area or variable table area becomes too large, FOCAL prints an error message (06.54 or 10.:5). The following routine allows the user to find out how many core locations are left for his use.

```
*ERASE
*FOR I=1,300; SET A(I)=1
?06.54                              (Disregard error code)
*TYPE %4,I*5,"LOCATIONS LEFT"
=  795LOCATIONS LEFT*
```

5.12 DEBUGGING

5.12.1 Using the Error Diagnostics

Whenever FOCAL detects an illegal command or improbable condition within a user's program, the execution of the program stops and an error message is printed in the form ?XX.XX@GG.ss, where ?XX.XX is the error message and GG.ss is the line at which the error occurred. (See Table 5-3 for the complete list of error messages.)

Depending upon the type of error detected, the user may ignore the error message or make program changes before continuing. For example, if the user types CTRL/C to terminate a loop, the error message ?01.00 is printed and program control goes to command mode; so, in this case, the user ignores the message and types his next command. If a program stops and the message ?03.05 is printed, the user must examine his program to determine which command line is wrong. In the following program, line 1.3 contains the instructions to transfer to a nonexistent line number.

```
*ERASE ALL
*1.1 SET A=2; TYPE "A",A,!
*1.2 SET B=4; TYPE "B",B,!
*1.3 GOTO 1.01
*1.4 TYPE "A+B",A+B
*GO
A=     2.0000
B=     4.0000
?03.05 @ 01.30
*
```

5.12.2 Using the Trace Feature

The trace feature is used to check the logic in part of a FOCAL program. To implement the trace feature, the user inserts a question mark (?) into a command string at any point. FOCAL prints each succeeding character as it is executed until another question mark is encountered or until the program returns to command mode. For example, the trace feature is used to print parts of 3 lines in the following program:

```
*ERASE ALL
*1.1 SET A=1
*1.2 SET B=5
*1.3 SET C=3
*1.4 TYPE ?A+B-C?,!
*1.5 TYPE ?B+A/C?,!
*1.6 TYPE ?B-C/A?
*GO
A+B-C=     3.0000
B+A/C=     5.3333
B-C/A=     2.0000*
```

NOTE

The WRITE command disables the trace feature.

5.13 FOCAL FUNCTIONS

The FOCAL functions improve and simplify arithmetic capabilities. In general, the FOCAL functions may be used anywhere a number or a variable is legal in a mathematical expression. A standard function call consists of three or four letters beginning with the letter F

and followed by an expression in parentheses. The FOCAL functions are summarized in Table 5-2.

The functions must be used with a legal FOCAL command. They cannot be used alone as commands. For example:

```
*SET Z=A+FSQT(FSIN(X))
```

Within a normal range of arguments, at least five significant digits of accuracy may be expected for the trigonometric, exponential, and logarithmic functions. The following functions are available to FOCAL users.

5.13.1 Sine Function (FSIN)

The sine function (FSIN) is used to calculate the sine of a user-specified angle in radians. The format for FSIN is:

FSIN (angle)

```
*TYPE FSIN(3.14159/4)
=    0.7071*
```

The format for calculation the sine of an angle in degrees is:

FSIN (degrees* 3.14159/180)

```
*TYPE FSIN(30*3.14159/180)
=    0.5000*
```

5.13.2 Cosine Function (FCOS)

The cosine function is used to calculate the cosine of a user-specified angle in radians. The format for FCOS is:

FCOS (angle)

```
*TYPE FCOS(2*3.14159),!
=    1.0000
*TYPE FCOS(.50000),!
=    0.8776
*TYPE FCOS(45*3.14159/180)
=    0.7071*
```

5.13.3 Exponential Function (FEXP)

The exponential function (FEXP) is used to compute e (e=2.71828) to a power specified by the user. The format for FEXP is:

FEXP (power)

```
*TYPE FEXP(1),!
=    2.7183
*TYPE FEXP(0),!
=    1.0000
*TYPE FEXP(-1-23456)
=    0.0000*
```

5.13.4 Logarithm Function (FLOG)

The logarithm function (FLOG) is used to compute the natural logarithm ($\log_e$) of a number specified by the user. The format for FLOG is:

FLOG (number)

```
*TYPE FLOG(1.0000),!
=    0.0000
*TYPE FLOG(1.98765),!
=    0.6870
*TYPE FLOG(2.065)
=    0.7251*
```

The following formulas are useful for finding logarithms to base 10:

$$\log_{10}X=\log_eX/\log_e10=0.434294\ \log_eX$$

5.13.5 Arctangent Function (FATN)

The arctangent function (FATN) is used to calculate the angle in radians of a user-specified tangent. The format for FATN is:

FATN (tangent)

```
*TYPE FATN(1.),!
=     0.7854
*TYPE FATN(.31305),!
=     0.3034
*TYPE FATN(3.14159)
=     1.2626*
```

5.13.6 Square Root Function (FSQT)

The square root function (FSQT) is used to compute the square root of an expression. The format for FSQT is:

FSQT (expression)

```
*TYPE FSQT(4),!
=     2.0000
*TYPE FSQT(9),!
=     3.0000
*SET Z=FSQT(144);TYPE Z
=    12.0000*
```

5.13.7 Absolute Value Function (FABS)

The absolute value function (FABS) is used to indicate the absolute (positive) value of an expression. The format for FABS is:

FABS (expression)

```
*TYPE FABS(-66),!
=    66.0000
*TYPE FABS(+23),!
=    23.0000
*TYPE FABS(-99.05)
=    99.0500*
```

5.13.8 Sign Part Function (FSGN)

The sign part function (FSGN) is used to output the sign part (+ or -) of a number with a value of 1. FSGN (0)=1. The format for FSGN is:

FSGN (expression)

```
*TYPE FSGN(6-4),!
=     1.0000
*TYPE FSGN(0),!
=     1.0000
*TYPE FSGN(-7)
=-    1.0000*
```

5.13.9 Integer Part Function (FITR)

The integer part function (FITR) is used to output the integer part of a number. The format for FITR is:

FITR (expression)

For positive numbers, FITR(X) is the greatest integer function. For negative numbers, FITR(-X) is the integer part of -X. The greatest integer function for negative numbers is obtained by FITR (-X)-1. For example:

```
*TYPE FITR(5.2),!
=     5.0000
*TYPE FITR(55.66),!
=    55.0000
*TYPE FITR(-4.1)
=-    4.0000*
```

5.13.10 Random Number Function (FRAN)

The random number function (FRAN) is used to generate non-statistical pseudo-random numbers in the range 0.5000 to 0.9999. No argument is used with the FRAN function. The format for FRAN is:

```
FRAN ( )
```

```
*TYPE FRAN(),!
=    Ø.6Ø73
*TYPE FRAN()
=    Ø.7376*
```

FRAN can be used to produce a less biased number. For example:

```
*SET A=FRAN()*5Ø
*SET B=A-FITR(A)
```

The value assigned to B is a random number in the range 0.0000 to 0.9999.

5.14 FOCAL OUTPUT OPERATIONS

The following is a description of symbols used in FOCAL output operations:

	Symbol	Explanation
To set output format,	TYPE %x.y	Where x is the total number of digits, and y is the number of digits to the right of the decimal point.
	TYPE %6.3, 123.456	FOCAL prints: =+123.456
	TYPE %	Resets output format to floating point.
To type symbol table,	TYPE $	Other statements may not follow on this line.

5.15 CONTROL CHARACTERS

FOCAL control characters and their explanations are shown below:

%	Output format delimiter	
!	Carriage return and line feed	
#	Carriage return	
$	Type symbol table contents	
( )	Parentheses	(mathematics)
[ ]	Square brackets	
< >	Angle brackets	
" "	Quotation marks	(text string)
??	Question marks	(trace feature)
SPACE key (names)		(nonprinting)
RETURN key (lines)		
ALT MODE key (with ASK statement)		
COMMA (expressions)		
Semicolon (commands and statements)		

TABLE 5-1

FOCAL COMMAND SUMMARY

Command	Abbreviation	Example of Form	Explanation
ASK	A	ASK X,Y,Z	FOCAL prints a colon for each variable; the user types a value to define each variable.
COMMENT	C	COMMENT	If a line begins with the letter C, the remainder of the line will be ignored.
CONTINUE	C	C	Dummy lines.
DO	D	DO 4.1	Execute line 4.1; return to command following DO command.
	D	DO 4.0	Execute all group 4 lines; return to command following DO command, or when a RETURN is encountered.
	DA	DO ALL	Execute all program lines until a RETURN is encountered.
ERASE	E	ERASE	Erases the symbol table.
		ERASE 2.0	Erases all group 2 lines.
		ERASE 2.1	Erases line 2.1.
		ERASE ALL	Erases all user input.
FOR	F	FOR i=x,y,z; (commands)	Where the command following is executed at each new value. x=initial value of i. y=value added to i until i is greater than z.
GO	G	GO	Starts indirect program at lowest numbered line number.

Table 5-1. (Cont.)

Command	Abbreviation	Example of Form	Explanation
GO?	G?	GO?	Starts at lowest numbered line number and traces entire indirect program until another ? is encountered, or until completion of program.
GOTO	G	GOTO 3.4	Starts indirect program (transfers control to line 3.4). Must have argument.

IF	I	IF (X)Ln,Ln,Ln
		IF (X)Ln,Ln; (commands)
		IF (X)Ln; (commands)
			Where X is a defined identifier, a value, or an expression, followed by three line numbers. If X is less than zero, control is transferred to the first line number. If X is equal to zero, control is to the second line number. If X is greater than zero, control is to the third line number.

LIBRARY CALL	#	LIBRARY CALL name	Calls stored program from the disk.
LIBRARY SAVE	LS	LIBRARY SAVE name	Saves program on the disk.
MODIFY	M	MODIFY n	Enables editing of any character on line n.
QUIT	Q	QUIT	Returns control to the user.
RETURN	R	RETURN	Terminates DO subroutines, returning to the original sequence.
SET	S	SET A=5/B*C;	Defines identifiers in the symbol table.

Table 5-1. (Cont.)

Command	Abbreviation	Example of Form	Explanation
TYPE	T	TYPE A+B-C;	Evaluates expression and prints = and result in current output format.
		TYPE A-B,C/E;	Computes and prints each expression separated by commas.
		TYPE "TEXT STRING"	Prints text. May be followed by ! to generate carriage return-line feed, or # to generate carriage return.
WRITE	W	WRITE	FOCAL prints the entire indirect program.
	W A	WRITE ALL	
		WRITE 1.0	FOCAL prints all group 1 lines.
		WRITE 1.1	FOCAL prints line 1.1.

TABLE 5-2

FOCAL FUNCTIONS

Function	Format	Interpretation
Square Root	FSQT(x)	Where x is a positive number or expression greater than zero.
Absolute Value	FABS(x)	FOCAL ignores the sign of x.
Sign Part	FSGN(x)	FOCAL evaluates the sign part only, with 1.0000 as integer.
Integer Part	FITR(x)	FOCAL operates on the integer part of x, ignoring any fractional part.
Random Number Generator	FRAN(x)	FOCAL generates a random number. The value of x is ignored.
[7]Exponential	FEXP(x)	FOCAL generates e to the power x. (2.71828^x)
[7]Sine	FSIN(x)	FOCAL generates the sine of x in radians.
[7]Cosine	FCOS(x)	FOCAL generates the cosine of x in radians.
[7]Arctangent	FATN(x)	FOCAL generates the arctangent of x in radians.
[7]Logarithm	FLOG(x)	FOCAL generates $\log_e$ (x).

[7]These are extended functions and may be chosen or deleted when FOCAL is loaded.

TABLE 5-3

FOCAL ERROR MESSAGES

Message	Explanation
?00.00	Manual start given from console.
?01.00	Interrupt from keyboard via CTRL/C.
?01.40	Illegal step or line number used.
?01.78	Group number is too large.
?01.96	Double periods found in a line number.
?01.:5	Line number is too large.
?01.;4	Group zero is an illegal line number.
?02.32	Nonexistent group referenced by DO.
?02.52	Nonexistent line referenced by DO.
?02.79	Storage was filled by push-down list.
?03.05	Nonexistent line used after GOTO or IF.
?03.28	Illegal command used.
?04.39	Left of = in error in FOR or SET.
?04.52	Excess right terminators encountered.
?04.60	Illegal terminator in FOR command.
?04.:3	Missing argument in display command.
?05.48	Bad argument to MODIFY.
?06.06	Illegal use of function or number.
?06.54	Storage is filled by variables.
?07.22	Operator missing in expression or double E.
?07.38	No operator used before parenthesis.
?07.:9	No argument given after function call.
?07.;6	Illegal function name or double operators.
?08.47	Parentheses do not match.
?09.11	Bad argument in ERASE.
?10.:5	Storage was filled by text.
?11.35	Input buffer has overflowed.
?20.34	Logarithm of zero requested.
?23.36	Literal number is too large.

Table 5-3. (Cont.)

Message	Explanation
?26.99	Exponent is too large or negative.
?28.73	Division by zero requested.
?30.05	Imaginary square roots required.
?31.<7	Illegal character, unavailable command, or unavailable function used.
?30.71	Undefined library command.
?30.>0	Bad argument or missing argument to library command.
?31.42	No such name in library directory.
?31.43	Attempt to enter a duplicate name in the directory.
?31.44	Library directory is full.

CHAPTER 6

FORTRAN

FORTRAN-D compiles and runs programs written in the PDP-8 version of FORTRAN II. Programs (usually created and stored with the Symbolic Editor) are compiled in a single pass and executed (automatically) immediately following compilation.

6.1 CALLING FORTRAN-D

To use FORTRAN-D, type:

```
.R FORT
```

FORTRAN requests the name of the input file, i.e., the file containing the FORTRAN program to be compiled and run. The user responds with the file name and the RETURN key. FORTRAN then requests the name of an output file in which to store the compiled version of the program. For normal usage, just the RETURN key need be typed. FORTRAN places the compiled code in a file of its own, then proceeds to run the program.

If a file name is entered for output, FORTRAN creates a permanent file in which the compiled binary program is saved. It is then possible to rerun this program without recompiling it. To run an already compiled program, call the FORTRAN operating system directly by typing:

```
.R FOSL
```

FOSL requests the name of an input file. Enter the name of the file containing the compiled binary. For example, if the user types:

```
•R FORT

 INPUT:MATRIX
OUTPUT:
```

FORTRAN compiles and executes the program MATRIX but does not save the compiled binary.

FORTRAN compiles and executes the program MATRIX and then leaves the compiled binary in the file named BMTRIX when the user types:

```
•R FORT

 INPUT:MATRIX
OUTPUT:BMTRIX
```

The FORTRAN binary program BMTRIX is executed without first being compiled when the user types:

```
•R FOSL

 INPUT:BMTRIX
```

All FORTRAN programs return to the Monitor when they have completed execution.

6.2 USING FORTRAN-D

Differing versions of PDP-8 FORTRAN offer slightly different features. FORTRAN-D differs in the way it is called into use (described above), and in its I/O capability (described below). FORTRAN-D allows three data formats:

I	Integer format
E	Exponential format
A	Alpha format, ASCII value of a character is stored as an integer value.

The standard device for READ and WRITE statements is the terminal, which is assigned device code 1. Because the terminal is so frequently used, FORTRAN-D includes two special input/output instructions, ACCEPT and TYPE. These instructions imply use of the terminal; therefore, the device code need not be specified. ACCEPT is especially convenient if data is to be entered at the keyboard because this instruction automatically supplies a line feed when the RETURN key is typed. Also, the user can correct an erroneously typed value by striking the RUBOUT key.

A FORTRAN-D program can also utilize the high-speed reader and punch for I/O. These devices are assigned code 2. Because the high-speed reader and punch are shared by all users, it is necessary to assign them if they are to be used. Assign the appropriate devices and mount tapes in the reader before running FORTRAN-D. An automatic DEASSIGN is performed by FORTRAN before it returns to the Monitor; therefore, the user must reassign the devices before each run.

FORTRAN-D also allows programs to read and write data files on the disk. These data files are completely separate from the program files. Data files are read and written by standard READ and WRITE statements within the FORTRAN-D program. The device code for the disk is 3. Because programs using the disk are treated differently by FORT (the FORTRAN-D compiler), it is necessary to identify programs which use the disk. These programs are identified by a DEFINE DISK statement as the first statement in any such FORTRAN-D program including a READ or WRITE statement with device code 3.

Just as FORT itself must ask for the name of its input and output files, so must a FORTRAN program ask for the names of its disk files. FORTRAN-D programs do this by typing INPUT: and OUTPUT: a second time. The user responds by typing the name of the files to be read or written by the program. FORTRAN-D asks for both input and output for all programs which include a DEFINE DISK statement. If

only input (or output) is to be used, the user responds to the other by typing the RETURN key.

6.3 LINE FORMAT

FORTRAN programs consist of a series of lines, each a string of 72 characters or less (the width of the terminal paper from margin to margin). Each line contains two fields: the first, which begins at the left margin, is an identification field; the second contains the statement field. Termination of a line is indicated to the computer by a carriage return, accomplished by typing the RETURN key.

The identification field can be blank, or can contain one of the following types of identification:

1. A Statement number. This number, which can be any positive integer from 1 to 2047 inclusive, identifies the statement on that line for reference by other parts of the program. Statement numbers are used for program control or to assist the programmer in identifying segments of his program.

2. The letter, C. This identifies the remainder of the line as a comment.

Although the identification field may be left blank, it cannot be omitted entirely. The statement field begins immediately after a blank space and extends through the next carriage return. A sample FORTRAN program is shown below:

```
C       THIS PROGRAM CALCULATES FACTORIALS
5       TYPE 200
10      ACCEPT 300,N
        IFACT=1
30      IF (N-1) 5,32,33
32      TYPE 400,N,IFACT
        GO TO 10
33      DO 35 I=1,N
        IFACT=IFACT*I
```

```
35      CONTINUE
        GO TO 32
200     FORMAT (/,"PLEASE TYPE A POSITIVE NUMBER",/)
300     FORMAT (I)
400     FORMAT (/,I," FACTORIAL IS",I)
        END
```

FORTRAN source programs are generated using the Symbolic Editor Program. The Editor will facilitate formatting lines by use of a tab character, permitting automatic movement to an indented second field.

6.3.1 Statement Numbers

Each statement can have a positive, nonzero integer (0-2047) as its number. The statement number is used to reference that particular statement elsewhere in a program. A statement number consists of one to four digits beginning at the left hand margin and is followed by a space or tab. Statement numbers can be assigned non-sequentially; however, no two statements can have the same number. There must be no more than 40 statement numbers in a given program, and they must have a value of 2047 (decimal) or less.

6.3.2 Statement Continuation Character

Frequently, a statement is too long to fit on one line. If the character single quote (') appears as the last character of a line before the carriage return, the next line is treated as a continuation of the preceding statement. A statement may be continued on as many lines as necessary to complete it, but the maximum number of characters in the statement cannot exceed 128 (about two formatted lines). For example:

```
10      A=B**2-(4.*A*C/(B**2+1.5*A*C))*4.3'
        +B**2+((SQTF(C)*SQTF(D))/(B**2+1.5*A*C))
```

is equivalent to the formula:

$$A = B^2 - \left(\frac{4 \cdot A \cdot C}{B^2 + 1.5 \cdot A \cdot C}\right) \cdot 4 \cdot 3 + B^2 + \left(\frac{\sqrt{C} \cdot \sqrt{D}}{B^2 + 1.5 \cdot A \cdot C}\right)$$

Although the continuation character, (') allows a single statement to extend over two or more lines, no more than one statement can be written on one line.

6.4 FORTRAN STATEMENTS

FORTRAN statements are of several types with various functions distinguished as follows:

1. Comment statements allow a programmer to insert notes within the program.
2. Arithmetic statements resemble algebraic formulas. They define calculations to be performed.
3. Control statements govern the sequence of statement execution within a program. These statements reference program line numbers.
4. Specification statements allocate data storage and specify input/output formats.
5. Input/output statements control the transfer of information into and out of the computer.

6.4.1 Comment Statements

The character C, at the left margin of a line, designates that line as a comment line. A comment has no effect upon the compilation process but can serve as a guide to program logic for later debugging, etc. There is no limit to the number of comment lines which can appear in a given program. A comment cannot be continued by use of the continuation character, ('), but must be continued in a second comment statement. For example:

```
C       THIS IS AN EXAMPLE
C       OF A COMMENT
```

6.4.2 Character Set

The following characters are used in the FORTRAN language:

1. The alphabetic characters, A through Z.
2. The numeric characters, 0 through 9.
3. The control characters:

; semicolon	CR carriage return
. period	LF line feed
' single quote	(left parenthesis
" double quote	) right parenthesis
, comma	

4. The operators:

** exponentiation	/ division
+ addition	* multiplication
- subtraction	= replacement

All other characters are ignored by the Compiler except as Hollerith information found in FORMAT statements (where all terminal characters are legal). The SPACE character has no grammatical function (it is not a delimiter) except in FORMAT statements and can be used freely to make a program easier to read.

6.4.3 Constants

Constants are explicit numeric values appearing in statements. Two types of constants, integer and real, are permitted in FORTRAN.

6.4.3.1 Integer Constants -- Integer (fixed-point) constants are represented by a string of one to four decimal digits, written with an optional sign and without a decimal point. An integer constant must fall within the range $\pm$2047. For example:

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

6.4.3.2 Real Constants -- Real constants are represented by a digit string, an explicit decimal point, and are written with an optional sign.[1] Real constants can also be written in exponential notation with an integer exponent to denote a power of ten (i.e., 7.2×10^3 is written 7.2E+3). A real constant may consist of any number of digits but only **the leftmost six** digits appear in the compiled program. Real constants must fall within the range 0.14×10^{-38} to 1.7×10^{38}. For example:

```
 +4.50   (plus sign is optional)
  4.50
-23.09
 -3.0E14 (same as -3.0 x 10^14)
  7.     (saved as 7.00000, not the same as the integer 7)
```

6.4.3.3 Fixed and Floating-Point Representation -- The difference between integers and real numbers in FORTRAN is the way in which each is represented in core memory. Both types of numbers are converted to binary to be stored in the computer.

A FORTRAN integer is stored in one 12-bit computer word. The sign of the number is kept in the high-order bit and the magnitude (the integer value) in the remaining 11 bits. This representation, shown schematically in Figure 6-1, is called fixed point, because the decimal point is always considered to be to the right of the rightmost digit. A FORTRAN integer may not exceed the range of $\pm$2047. All integers greater than $\pm$2047 are taken modulo 2048 (i.e., 2049 is considered to be 1; 4099, to be 3).

---

[1] Where a number is to be identified as being negative, a minus sign (-) must be used. A plus sign (+) is optional; with no sign, a number is considered positive.

The floating-point format consists of two parts: an exponent (or characteristic) and a mantissa. The mantissa is a binary fraction with the radix point assumed to be to the left of bit one of the mantissa. The mantissa is always normalized; meaning it is stored with leading zeros eliminated so that the leftmost bit is always significant. The exponent represents the power of two by which the mantissa is multiplied to obtain the true value of the number for use in computation. The exponent and mantissa are both stored in two's complement form.

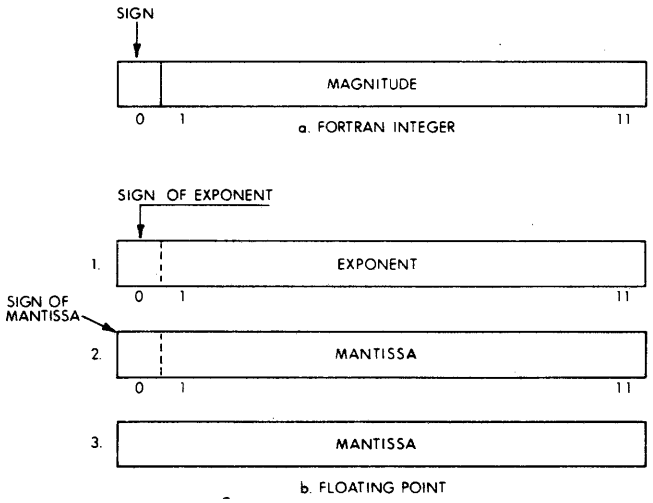


Figure 6-1. Number Representation

Users should not attempt to input floating point constants of more than six decimal digits, either in the FORTRAN source program or via the run-time ACCEPT statement.

Integers cannot appear in floating-point expressions except as exponent or subscripts. Some examples of illegal and legal expressions are as follows:

Expression	Legal	Mode
A(I)*B(J)**2	Yes	Floating
I(M)*K(N)	Yes	Fixed
16.*B	Yes	Floating
(K+16)*3	Yes	Fixed
A**(1+2)/B	Yes	Floating
8*A	No	-
4.*J	No	-
I+D	No	-

6.4.4 Variables

A variable is a symbol whose value may change during execution of a program. The name of a variable consists of one or more alphanumeric characters, the first of which must be alphabetic. Only the first four characters are interpreted as defining the variable name; the rest are ignored. For example, the name EPSILON would be interpreted by the Compiler as EPSI. Since only the first four characters are meaningful, the two names XSUM1 and XSUM2 would be considered identical.

Spaces, as mentioned earlier, are ignored. The name EX IT represents one variable, not two. Thus, EX IT, EXIT, or even EXI T are identical names as far as the Compiler is concerned, and they all refer to the same numerical quantity.

The type (or mode) of a variable (integer or real) is determined by the first letter of the variable name.

1. Integer variables begin with the letters:

   I, J, K, L, M, or N

2. Real variables begin with any letter other than those above.

Variables of each type may be either scalar or array variables, as explained below.

6.4.4.1 Integer Variables -- Integer variables undergo arithmetic calculations with automatic truncation of any fractional part. For example, if the current value of K is 5 and the current value of J is 9, then J/K would yield 1 as a result.

6.4.4.2 Real Variables -- A variable is real when its name begins with any character other than I, J, K, L, M, or N. Real variables undergo no truncation in arithmetic calculations. Real variables may be converted to integer variables, and vice-versa, across an equal sign.

6.4.4.3 Scalar Variables -- A scalar variable, which may be either integer or real, represents a single number, as opposed to an array (below) representing a collection of numbers. For example, the following are scalar variables:

```
LM
G2
A
TOTAL   (considered to be TOTA by the computer)
J
```

6.4.4.4 Array Variables -- An array variable represents a single element of a one-dimensional array of quantities. The variable is denoted by the array name followed by a subscript enclosed in parentheses. The subscript may be any combination of integer variables and integer constants forming a valid expression, as follows:

(V) (V+C) (V-C) (C)

where V is a fixed point (integer) variable, and C is a fixed-point constant (not equal to 0).

The value of the expression in parentheses determines the referenced array element. For example, the row matrix, A_1, would be represented by the subscripted variable A(1), and the second element in the row would be represented by A(2). Examples of array variables are:

Legal Forms	Illegal Forms
Y(1)	A(2+1)
A(K+2)	B(C)

6.4.5 DIMENSION Statement

Array names must be identified as such to the FORTRAN Compiler. Two items of information must be provided in any program using arrays:

1. Which are the subscripted variables?
2. What is the maximum value of the subscript? (When an array is used, a certain amount of storage space must be set aside by the Compiler for the array elements.)

This information is provided by the DIMENSION statement:

```
DIMENSION A(20),B(15)
```

where A and B are array names, and the integer constants 20 and 15 are the maximum dimensions of each subscript.

The rules governing the use of array variables and the DIMENSION statement are as follows:

1. All array names must appear in a DIMENSION statement.
2. DIMENSION statements may be used more than once and may appear anywhere in the FORTRAN program, provided that the DIMENSION of an array appears before any statement which references the array.
3. Any number of arrays can be defined in a single DIMENSION statement.

4. For notes on how to implement double subscripts (i.e., A(I,J)), see Section 6.8, Implementation Notes.

Array variables may be either integer or real, depending upon the initial letter of the array name.

```
DIMENSION LIST(30),MAT(100),REGR(20)
```

In the statement above, the names LIST and MAT designate integer arrays; that is, all elements of both arrays are integers. The third name, REGR, designates a floating point, or real array. The first array is a list containing a maximum of 30 elements; the second array has a maximum of 100 elements.

The third array is a floating-point array and there are a maximum of 20 elements in it. Not all elements of an array need be used in the course of a program; but, if using the DIMENSION statement the variable LIST (31) could not be referenced without the occurrence of an error message.

6.5 FORTRAN ARITHMETIC

6.5.1 Arithmetic Operators

The arithmetic operators are symbols representing the common arithmetic operations. The important rule about operators in the FORTRAN arithmetic expressions is that: every operation must be explicitly represented by an operator. In particular, the multiplication sign must never be omitted. A symbol for exponentiation is also provided since superscript notation is not available on a terminal.

Normally, a FORTRAN expression is evaluated from left to right, like an algebraic formula. There are exceptions to this rule; certain operations are always performed before others, regardless of order. This priority of evaluation is as follows:

1.	Expressions within parentheses	( )
2.	Unary minus	-
3.	Exponentiation	**
4.	Multiplication or Division	* or /
5.	Addition or Subtraction	+ or -

The term "binding strength" is frequently used to refer to the relative position of an operator in a table such as the one above, which is in order of descending binding strength. Thus, exponentiation has a greater binding strength than addition, and multiplication and division have equal binding strength.

The unary minus is the arithmetic operator which indicates that a quantity is less than zero, such as -53, -K, -12.3. It refers only to the constant or variable which it precedes as opposed to a binary operator, which refers to operands on either side of itself as in the expression A-B. A unary minus is recognized by the fact that it is preceded by another operator, not by an operand. For example:

```
A+B**-2/C-D
```

The first minus sign (indicating a negative exponent) is unary; the second (indicating subtraction) is binary. At present it is not possible to raise an integer variable or integer constant to an integer value with FORTRAN-D. Only real values can be raised to integer powers.

The left-to-right rule can be stated more precisely: A sequence of operations of equal binding strength is evaluated from left to right. To change the order of evaluation, parentheses are required. Thus, the expression A-B*C is evaluated as A-(B*C), not (A-B)*C. Examples of the left-to-right rule follow:

The expression:	Is evaluated as:
A/B*C	(A/B)*C
A/B/A	(A/B)/C
A**B**C	(A**B)**C

6.5.1.1 Use of Parentheses -- Note the use of parentheses in the example below. They are used to enclose the subscript of the dimensional variable, D; to specify the order of operations of the expression involving A, B, and C; and to enclose the argument of the function SINF.

```
D(I)+(A+B)**C+SINF(X)
```

In algebra there are several devices, such as square brackets [ ], rococo brackets { } , etc., for distinguishing between levels when expressions are nested. In FORTRAN, only the parentheses are available, so the programmer must be especially careful to pair parentheses properly. In any given expression, the number of left parentheses must be equal to the number of right parentheses.

An easy way to check the proper pairing of parentheses is by counting out, illustrated in the following example:

```
(Z+AM*  (AM+1.))/((X**2+C**2)*P)
1       2     10 12          1  0
```

The procedure is this: Reading the expression from left to right, assign the number, 1, to the first left parenthesis (if you encounter a right parenthesis first, the expression is already wrong). Increase the count by one each time a left parenthesis is read, and decrease the count by one when a right parenthesis is found. When the expression has been completely scanned, the count should be zero. If it becomes less than zero during the scanning, there are too many right parentheses. If it is greater than zero at the end of an expression, there are excess left parentheses.

6.5.2 Arithmetic Expressions

An algebraic formula such as:

$$5a + 4b(x^2 - x_0)$$

represents a relationship between symbols (a, b, x, x_0) and constants (5, 4, 2) indicated by mathematical functions and arithmetic operators (+, -, multiplication, exponentiation). This same formula can be written as a FORTRAN arithmetic expression with very little change in appearance:

```
(5.*A+4.*B*(X**2-XZRO))
```

The construction of both expressions is the same; the differences are notational.

Elements of an arithmetic expression are of four types: constants, variables, operators, and functions. An expression may consist of a single constant or variable or a string of constants, variables, and functions connected by operators.

Examples of arithmetic expressions follow; each expression is shown with its corresponding algebraic form.

Algebraic Expression	FORTRAN Expression
$\frac{2\sqrt{x}}{3}$	`2.*SQTF(X)/3.`
$\frac{3x\pi - 2(x+y)}{4.25}$	`(3.*X*PI-2.* (X+Y) ) /4.25`
$a°\sin\theta + 2a\cos(\theta - 45)$	`A*SINF(THTA)+2.*A*COSF ((THTA)-0.78540)`

$$\frac{(a^2 - b^2)}{(a + b)^2}$$

```
(A**2-B**2) / (A+B)**2
```

6.5.3 Arithmetic Statements

The arithmetic statement relates a variable, V, to an arithmetic expression, E, by means of the replacement operator, (+):

```
V=E
```

Such a statement looks like a mathematical equation, but is treated differently. The equal sign is interpreted in a special sense; it does not represent a relationship between left and right members, but rather specifies an operation to be performed.

In an arithmetic statement, the value of the expression to the right of the equal sign replaces the value of the variable on the left. This means that the value of the left-hand variable will change after the execution of an arithmetic statement. A few illustrations of arithmetic statements are given below.

```
1.   VMAX = VO + AXT
2.   T = 2.*PI*SQTF(1./G)
3.   PI = 3.14159
4.   THTA = OMGA + ALPH*T**2/2.
5.   MIN = MIN0
6.   INDX = INDX + 2
```

With the interpretation of the equal sign stated above, Example 6 becomes meaningful as an arithmetic statement. If, for example, the value of INDX is 40 before the statement is executed, its value will be 42 after execution.

In arithmetic expressions, a binary operator requires an operand on its left and right. The equal sign of an arithmetic statement is also considered to be a binary operator, as demonstrated in the following revised table of operators:

Operator	Use	Interpretation
- (Unary)	-A	negate A
**	A**B	raise A to the Bth power
*	A*B	multiply A by B
/	A/B	divide A by B
+	A+B	add B to A
- (Binary)	A-B	subtract B from A
=	A=B	replace the value of A with the value of B

The replacement operator is considered to have the lowest binding strength of all operators; therefore, the expression on the right is evaluated before the operation indicated by the equal sign is performed.

6.5.3.1 Multiple Replacement -- An important result of treating the equal sign as an operator is that operations can be performed in sequence. Just as there can be a series of additions, A+B+C, there can also be a series of replacements:

```
A=B=C=D
```

Notice that because the operand to the left of an equal sign must be a variable, only the rightmost operand, represented by D in the example, may be an arithmetic expression. The statement is interpreted as follows: "Let the value of the expression D replace the value of the variable C, which then replaces the value of the variable B" and so on. In other words, the value of the rightmost expression is given to each of the variables in the string to the left. A common use for this construction is in setting up initial values:

```
XZRO=SZRO=AZRO=0.
T=T1=T2=T3=60.
P=FP=4.*ATM-AK
```

Only simple variables will compile correctly in this manner. For example, statements of the type A(1)=A(2)=R(1)=0.123 are not allowed and will not compile properly (subscripted variables may not be used in multiple replacement statements).

Multiple replacement done in a single statement must not contain mixed mode variables. That is;

A=B=C=10.	compiles correctly
I=J=7	compiles correctly
A=J=7	does not compile correctly

Mode Conversion

Another useful result in treating the equal sign as an operator is that the value of an expression on the right of an equal sign is converted to the mode of the left-hand variable, if necessary, before storage. For example:

A=M Stores the value of M as a floating-point number in A

K=B Stores the value of B (truncated) as an integer number in K

If B = 4.75 and M = 7, the conversion above will result in the following values being assigned:

A = 7.00000

K = 4

6.5.4 Functions

Functions are used in FORTRAN just as they are in ordinary mathematics, acting as variables in arithmetic expressions. The function name represents a call to a special subprogram which performs the calculations to evaluate the function; the result is used in the computation of the expression in which the function occurs. FORTRAN-D provides several mathematical functions: square root, sine, cosine, arctangent, exponentiation, and natural logarithm.

The argument of a function can be a simple variable, a subscripted variable, or an expression. The argument must be in a floating-point format. FORTRAN recognizes a symbol as a function when it is a predefined symbol ending in F and followed by an argument enclosed in parentheses (if the F is missing from the term, the symbol is treated as a subscripted variable). The argument of a function can consist of another function or groups of functions. For example, the expression:

```
LOGF(SINF(X/2.)/COSF(X/2.))
```

is equivalent to $\log \tan \left(\frac{X}{2}\right)$

FORTRAN-D contains the following functions:

Function Name	Meaning
ATNF(X)	Arctangent X, where X is expressed in radians
COSF(X)	Cosine of X, where X is expressed in radians
EXPF(X)	Exponential of X
LOGF(X)	Logarithm of X
SINF(X)	Sine of X, where X is expressed in radians
SQTF(X)	Square root of X

6.6 PROGRAM CONTROL STATEMENTS

In this section, FORTRAN statements are discussed in the context of program sequences. FORTRAN statements are executed in the order in which they are written unless instructions are given to the contrary by use of the program control statements. These statements allow the programmer to alter sequence, repeat sections, suspend operations, or bring the program to a complete halt.

6.6.1 END Statement

END occurs alone on a line and indicates the physical end of the program to the FORTRAN Compiler. It can be preceded by a line number. Every program must contain an END statement.

6.6.2 STOP Statement

A program arranged so that the last written statement is the final and only stopping place needs no other terminating indication; the END statement automatically determines the final halt. Many programs, however, contain loops and branches so that the last executed statement can be somewhere in the middle of the written program. Frequently there is more than one stopping point. Such terminations are indicated by the STOP statement. This causes a final, complete halt; no further computation is possible, although the program may be completely restarted from the beginning.

When a STOP is encountered during program execution, the system signifies that a STOP has occurred by outputting an exclamation point (!) to the terminal or high-speed punch, whichever is being used as the output device.

6.6.3 PAUSE Statement

The STOP statement prevents further computation after it has been executed. There is a way, however, to suspend operation for a time and then restart the program. This procedure is frequently necessary when the user must do such tasks as loading and unloading paper tapes in the middle of a program. This kind of temporary halt is provided by the PAUSE statement. The PAUSE statement halts the program and returns control to the EduSystem 50 Monitor. The user may then perform any necessary manipulations and restart the program by typing the Monitor command START.

6.6.4 GO TO Statement

There are various ways in which program flow may be directed. As shown schematically in Figure 6-2, a program may be a straight-line sequence (1), or it may branch to an entirely different sequence (2), return to an earlier point (3), or skip to a later point (4). The blocks represent sections of FORTRAN code. The lines indicate the path which control takes as the program executes.

All of these branches can be performed in several ways, the simplest of which uses the statement:

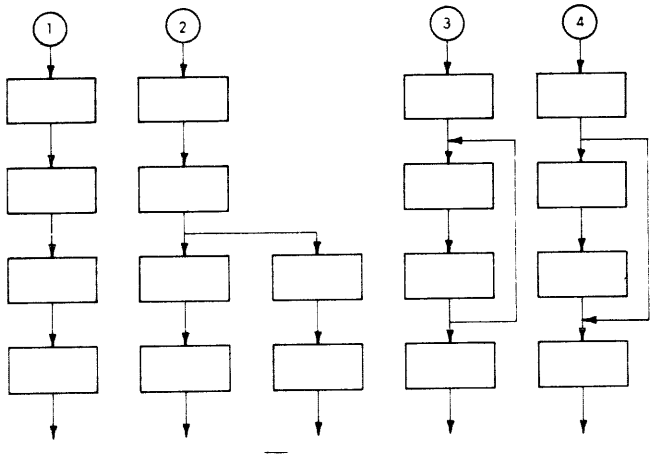


Figure 6-2. Program Flow

```
GO TO n
```

where n is a statement number in the program. The use of this statement is described in the following example, which also illustrates the construction of a loop, the name given to program branches of the type shown in the example above.

6.6.5 Example of Integer Summation

In the following example, the sum of successive integers is accumulated by repeated addition. The main computation is provided by the three-instruction loop beginning with Statement 2. The statements preceding this loop provide the starting conditions, called the initialization. The partial sum is set to zero, and the first integer is given the value of one. The loop then adds the integer value to the partial sum, increments the integer, and repeats the operation.

```
C       SUM OF FIRST N INTEGERS BY ITERATION
        KSUM=0
        INUM=1
2       KSUM=INUM+KSUM
        INUM=INUM+1
        GO TO 2
        END
```

6.6.6 IF Statement

The program shown in the preceding example performs the required computation, but note that the loop is endless. To get out of the loop the user must know when to stop the iteration and what to do afterwards.

The IF statement fills both requirements. It has the following form:

```
IF(E) K,L,M
```

where E is any variable name, arithmetic expression, or arithmetic statement, and K, L, and M are statement numbers. The statement is interpreted in this way:

```
If the value of E < 0,GO TO statement K
                E = 0,GO TO statement L
                E > 0,GO TO statement M
```

Thus, the IF statement decides when to stop a loop by evaluating an expression. It also provides program branch choices with the transfer of control, depending on the results of the evaluation of E. For example:

```
C       SUM OF THE FIRST 50 INTEGERS
        KSUM=0
        INUM=1
2       KSUM=INUM+KSUM
        INUM=INUM+1
        IF (INUM-50)2,2,3
3       STOP
        END
```

In the foregoing example, the initialization and main loop are the same as for the example in Figure 6-2 except that the GO TO statement of the earlier program has been replaced by an IF statement. The IF statement says, "if the value of the variable INUM is less than, or equal to, 50 (which is the same as saying that if the value of the expression INUM-50 is less than or equal to zero), transfer control to Statement 2 and continue the computation. If the value is greater than 50, stop." (See Section 6.8, Implementation Notes, for an alternate solution.)

A loop may also be used to compute a series of values. The following illustration is an example of a program to generate terms in the Fibonacci series of integers, in which each succeeding member of the series is the sum of the two members preceding it:

```
C       FIBONACCI SERIES, 100 TERMS
        DIMENSION FIB(100)
        FIB(1)=1.0
        FIB(2)=1.0
        K=3
5       FIB(K)=FIB(K-1)+FIB(K-2)
6       K=K+1
        IF (K-100)5,5,10
10      STOP
        END
```

In this program, the initialization includes a DIMENSION statement which reserves space in storage, and two statements which provide the starting values necessary to generate the series. Each time a term is computed, the subscript is incremented so that each succeeding term is stored in the next location of the table. As soon as the subscript is greater than 100, the calculation stops.

6.6.7 DO Loops

Iterative procedures such as the program loop are so common that a more concise way of presenting them is warranted. Three statements are required to initialize the subscript, increment it, and test for termination. The following type of statement combines all these functions:

```
DO n J=K1,K2,K3
```

here n is a statement number, J is a simple (non-subscripted) integer variable, and K1, K2, and K3 are simple integer variables or integer constants which provide, in order, the initial value to which J is set, the maximum value of J for which the loop will be executed, and the amount by which J is incremented at each return to the beginning of the loop. If K3 is omitted from the statement it is assumed to be one (1). Statement n must be a CONTINUE statement.

```
C       FIBONACCI SERIES, 100 TERMS
        DIMENSION FIB(100)
        FIB(1)=1.0
        FIB(2)=1.0
        DO 5 K=3, 100
        FIB(K)=FIB(K-1)+FIB(K-2)
5       CONTINUE
        STOP
        END
```

In words, the DO statement says "Execute all statements through Statement 5 with K=3; when Statement 5 is encountered, perform the following test: If K+1 is less than or equal to 100, set K=K+1 and continue the program by executing the first statement after the DO statement. If K+1 is greater than 100, the next sequential statement following Statement 5 is executed."

DO loops are commonly used in computations with subscripted variables. In such cases, it is usually necessary to perform the loops within loops. Such nesting of DO loops is permitted in FORTRAN.

```
C       FIRST LOOP
        DO 10 I=1,20
        X(I)=0.
C       NESTED LOOP FOLLOWS
        DO 5 K=2,40,2
        X(I)=X(I)+(B(K)-Z(K))**2
5       CONTINUE
C       END OF NESTED LOOP
        A(I)=X(I)**2+C(I)
10      CONTINUE
```

Sequential elements in the array X(I) are formed by summing the square of the difference of every second element in the B and Z arrays. Then the array A(I) is formed by summing every element in the array C(I) and the square of every element in the array X(I). The algebraic expression for the loop is as follows:

$$A_i = x_i^2 \quad C_i \qquad \text{for } i=1,2,3,\ldots 20$$

where

$$x_i = \sum_{k=2}^{40} (b_k - z_k)^2 \qquad \text{for } k=2,4,6,\ldots 40$$

The following three rules loops must be observed:

1. DO loops may be nested, but they may not overlap. Nested loops may end on the same statement, but an inner loop may not extend beyond the last statement of an outer loop. Figure 6-3 schematically illustrates permitted and forbidden arrangements.
2. If the user transfers into the range of a DO loop, the value to be incremented (J, for example) is not automatically initialized as specified in the DO statement. Transferring into the range of a DO loop is allowed as long as:
   a. Control was originally transferred out of the DO loop by some means other than by completing it.
   b. Incrementing and testing start with the current value of J at the time control returns to the loop.
3. A DO loop must end on a CONTINUE statement.

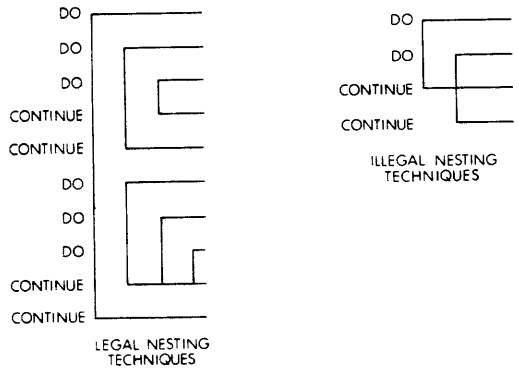


Figure 6-3. Legal and Illegal Nesting Techniques

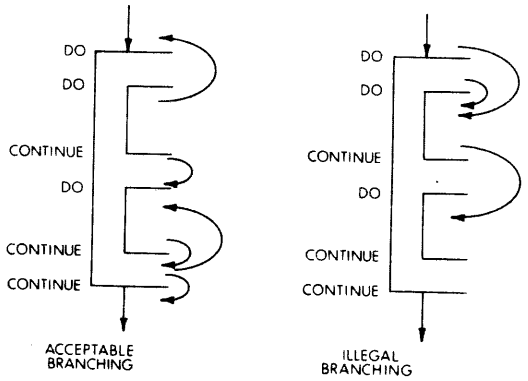


Figure 6-4. Program Branching in DO Loops

6.6.7.1 CONTINUE Statement -- A special statement (CONTINUE) is provided which is not an executable statement itself, but provides a termination for all DO loops. DO loops must be terminated on a CONTINUE statement. The CONTINUE statement is identified with the line number given in the DO statement. For example:

```
        DO 37 MM=1,10
        IF (X(MM)-100.) 37,42,37
37      CONTINUE
        GO TO 102
42      STOP
```

A single CONTINUE statement can be referenced more than once in a single DO loop or can serve as the terminating line for two or more nested DO loops.

6.6.8 Computed GO TO

The GO TO statement previously described is unconditional and provides no alternatives. The IF statement offers a maximum of three branch points. One way of providing a greater number of alternatives is by using the computed GO TO, which has the following form:

```
GO TO (K1,K2,K3,...,Kn),J
```

where K_n is a statement number, and J is a simple integer variable, which takes on values of 1,2,3,...n according to the results of some previous computation. For example:

```
IVAR=14*J/2+K
GO TO (5,7,5,7,5,7,10),IVAR
```

causes a branch to Statement 5 when IVAR=1, 3, or 5; to Statement 7 when IVAR=2, 4, or 6; and to Statement 10 when IVAR=7. When IVAR is less than 1 or greater than 7, the next sequential statement after the GO TO is executed.

6.7 FORTRAN INPUT/OUTPUT

So far, we have assumed that all information (programs, data, and sub-programs) is in memory, without regard to how it is put there. Programs are read by a special loader, but the programmer is responsible for the input of data and the output of results by including directions for I/O operations in his program.

For any input/output procedure, several questions must be answered:

1. In which direction is the data going? The data coming in is being read into memory; information going out is being written on whatever medium is specified.
2. Which device is being used? Information may be transferred between core and whatever input/output devices are available; each I/O operation must specify the device involved.
3. Where in core memory is the data coming from or going to? The amount of data and its location in the computer storage must be specified.
4. In what mode is the data represented? In addition to floating and fixed-point modes for numeric data, there is the Hollerith mode for transferring alphanumeric or text information.
5. What is the arrangement of the data? The format of incoming or outgoing data must be specified.

For every data transfer between core memory and an external device, two statements are required to provide all of the information listed above. The first three items are specified by the input/output statement, and the last two items are determined by the FORMAT statement.

6.7.1 Data Formats

FORTRAN-D provides for communication of data to and from a program in the following ways:

6.7.1.1 ASCII Coded Data -- The terminal can be used to transfer data to the program either via the keyboard (in which case the user types the data) or from previously punched paper tape (read via the terminal tape reader). Data can be output from a program to the terminal producing a printed copy with or without the corresponding punched paper tape. The high-speed reader and punch can also be used for data transfer via punched paper tape. No printed copy is made when output is to the high-speed punch.

6.7.1.2 Binary Coded Data -- System disk can also be used for data transfer, in which case the data is stored as a core image. Integers are read and written as single 12-bit words, floating-point numbers as three words. Alphanumeric information is transmitted as 8-bit ASCII coded characters right-justified in 12-bit words (one character per word).

6.7.2 Input/Output Statements

Input/Output statements control the transfer of information. As illustrated below, I/O statements consist of three basic items of information: the device being accessed and the direction of transfer, the number of the FORMAT statement controlling the arrangement of data, and the list of variable names whose values are to be output or changed by new input.

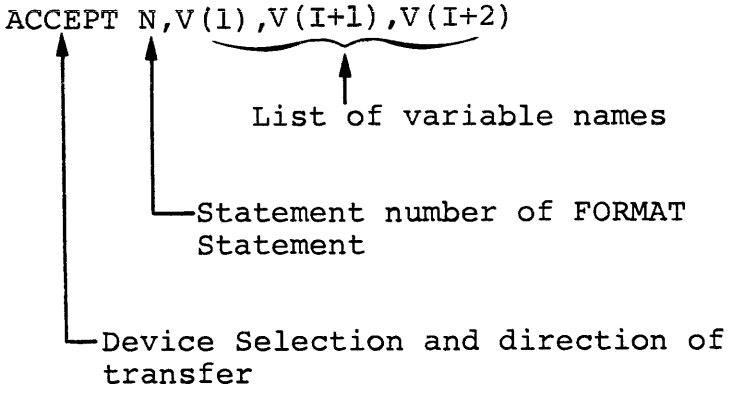
NOTE

There is a restriction on subscripted variables when used with I/O statements. Subscripts to be used with I/O statements must be of the form: LL, where each L is a letter, and not of the form LD, where D is a digit. For example:

```
DO 10 L1=1,4
ACCEPT 7,A(L1)
```

will not store information correctly. The statement should read:

```
DO 10 LL=1,4
ACCEPT 7,A(LL)
```



6.7.2.1 ACCEPT and TYPE Statements -- ACCEPT and TYPE transfer information between the terminal and EduSystem 50. ACCEPT causes information to be read into core memory from either the keyboard, the terminal paper tape reader, or the high-speed reader if it is assigned before calling FORTRAN-D. ACCEPT is especially convenient if data is to be entered at the keyboard since it automatically supplies line feed when the RETURN key is typed. Also, the user may correct an erroneously typed value by typing the RUBOUT key.

TYPE causes information to be transferred from core memory to the terminal printer, the terminal paper tape punch, or the high-speed punch if it is assigned before calling FORTRAN-D.

If the user needs the high-speed reader and punch for I/O, he must assign the devices for his use before calling the FORTRAN compiler (FORT) or operating system (FOSL). Once logged into EduSystem 50, he replies to Monitor's dot with the appropriate assign statements. For example:

```
•ASSIGN P
P ASSIGNED

•ASSIGN R
R ASSIGNED
```

The Monitor replies with P ASSIGNED and R ASSIGNED in response to the user. If the device requested is not available (being used by someone else), the Monitor responds with a message telling who has the device assigned. The high-speed reader/punch is device code 2. If running several programs, the user should reassign the devices before each run.

6.7.2.2 READ and WRITE Statements -- EduSystem 50 FORTRAN also allows programs to read and write data files on disk. These data files are completely separate from program files. Data files are read and written by standard READ and WRITE statements within the FORTRAN program. The device code for the disk is 3. Since programs which use disk are treated differently by FORT from those which do not use disk, it is necessary to identify programs which do. This is done by placing a DEFINE DISK statement as the first statement in any FORTRAN program which includes a READ or WRITE statement with a device code of 3.

Just as FORT must ask for the names of its input and output files, so must a FORTRAN program ask for the names of its disk files. FORTRAN programs do this by typing INPUT: and OUTPUT: a second time. The user responds by typing the names of the data files to be read or written by the program. FORTRAN will ask for

both INPUT and OUTPUT for all programs which include a DEFINE DISK statement. If only one is to be used, respond to the other by typing the RETURN key.

6.7.3 Variable Specification in I/O Statements

Following the instruction that selects the device and direction of transfer is the statement number of the FORMAT statement that controls the arrangement of the information being transferred. For example:

```
        ACCEPT 10,A
10      FORMAT (E)
```

Every I/O statement must have a reference to a FORMAT statement.

The final item specified in the I/O statement is the list of variables. This is a sequential list of the names of variables and array elements whose values are to be transferred in the order indicated. There is no restriction on the number of names which may appear in the list of an I/O statement, as long as the total statement length does not exceed 128 characters. The modes of the variables named need not agree with the corresponding FORMAT statement; however, the modes specified in the FORMAT statement take precedence. For example, where A=3.2,J=27,KAL=302, and BOB=7.58:

```
        TYPE 23,A,J,KAL,BOB
23      FORMAT (I,E,I,E)
```

The decimal portion of A is dropped and the number 3 is printed as an integer; the value of J is printed as a normalized number; KAL is printed as an integer; and BOB is printed as a normalized number. The output would look like the following:

```
 3      0.270000E+2      302    0.758000E+1
```

NOTE

In READ and ACCEPT statements, although the number is read according to the FORMAT statement, it is stored according to the mode of the variable. For example:

```
5       ACCEPT 10,A
10      FORMAT (I)
```

causes the number 12.3 typed by the user to be read as 12 and stored as 0.120000E+2.

Array names included in I/O lists must be subscripted in one of the following forms:

A(V) A(V+C) A(V-C) A(C)

where A is the array name, V is a simple integer variable and C is a positive nonzero integer constant.

```
        TYPE 10,A,I,B,C(I+1),N(J+1)
10      FORMAT (E,I,E,/)
```

If the list contains more names than there are elements in the FORMAT statement, the FORMAT statement is reinitialized when the elements are exhausted. The first element in the FORMAT statement then corresponds to the next name in the list. For instance, in the preceding example when the value of the variable, B, is printed in the E format, the control character, slash (/), causes a carriage return/line feed to occur. Then the FORMAT statement is reinitialized, and the array element, C(I+1), is printed in the E format and the array element N(J+1) in the I format.

The list does not have to exhaust the elements of a FORMAT statement. If there are fewer names in the list than there are elements in the FORMAT statement, the program completes the I/O operation and proceeds to the next sequential FORTRAN statement. If this next statement is another I/O statement that references a previously unexhausted FORMAT statement, that FORMAT statement is reinitialized. FORMAT statements are reinitialized when they are referenced or when all of their elements are exhausted.

6.7.4 FORMAT Statement

The FORMAT statement controls the arrangement and mode of the information being transferred. The values of names appearing in the I/O statement list are transferred in the mode specified by the corresponding element in the FORMAT statement. These controlling elements consist of the characters E, I, A, slash (/), and quote ("). The set of elements must be enclosed in parentheses and separated by commas. For example:

```
FORMAT (A,E,I,/,"HOLLERITH")
```

The control elements E and I are used for defining the mode of the data being transferred. When a variable is transferred in the E format, it is stored or output in floating-point form. If the variable is transferred in the I format, it is stored or output in fixed-point or integer form. Mode conversion on input or output can be accomplished because the elements in the FORMAT statement define the mode of the data. The mode of the original variable is overridden where necessary. For example:

```
      TYPE 10,A
10    FORMAT (I)
```

The variable, A, is printed as an integer, and the fractional part of A is truncated. If A has a value of 14.96, only the integer part, 14, is printed. If A has an absolute value of less than one, zero is printed.

6.7.5 The Format Specification

The control element, A, is used for defining the alphanumeric mode of data I/O. When a variable is to be assigned an alphanumeric value, data is read one character per variable. FORTRAN ignores CTRL/C, blank tape, RUBOUT, and 0200 code (leader/trailer tape). FORTRAN does not see the form-feed character when input is from the disk. The decimal equivalent of the ASCII value of the character is assigned to the variable. For example:

A = 301 (OCTAL) = 192 (decimal)

Any variable assigned the alphanumeric value, A, would be set equal to 192.

It is possible to do arithmetic with integer variables assigned alphanumeric values. For example:

```
      DO 10 J=1,5
      ACCEPT 12,K(J)
      IF (K(J)-141) 10,40,10
10    CONTINUE
12    FORMAT(A)
```

where the IF statement tests to see if the last character read is a carriage return (which is ASCII 215 or 141 decimal); if so, control transfers to Statement 40; if not, control stays within the DO loop.

It is not possible to do arithmetic with real variables assigned alphanumeric values. Output in alphanumeric format converts the value of the variable into an ASCII character and prints that character. For example:

```
12      FORMAT (A)
        DO 20 I=1,5
        TYPE 12,A(J)
20      CONTINUE
```

If the variables A(1) through A(5) were not originally assigned alphanumeric values, the results of the output can be meaningless.

6.7.6 Input Formats

Input data words can only consist of a sign, a decimal value, an exponent value if the data is floating-point, and a field terminating character such as space. Any character that is not a number, decimal point, sign, or E can be used to terminate a field except the RUBOUT character. When typing data, any number of spaces or other nonnumeric characters can be typed before the sign or decimal value in order to make the hard copy more readable.

Input data can be transferred into core memory from either the terminal paper-tape reader, the keyboard, or the high-speed reader. Input can be entered in either fixed- or floating-point modes (integers or decimal numbers). The mode in which data is stored in core memory is controlled by the first letter of the variable name. The characters read into core are determined by the corresponding element in the FORMAT statement.

6.7.6.1 Integer Values--the I Format -- An integer data field consists of sign[2] and up to six decimal characters. Some examples of integer values are as follows:

[2] Plus sign can be represented by a plus or space character. Minus is represented by a minus character. If a sign character is absent from the data word, the data is stored as positive.

Typed Numbers	Values Accepted
-2001	-2001
-40	-0040
-0040	-0040
16	0016
+2041	2041

6.7.6.2 Real Values--the E Format -- A floating-point input word consists of a sign, the data value up to six decimal characters, an E if an exponent is to be included, the sign of the exponent, and the exponent (i.e., the power of ten by which the data word is multiplied). For example:

ddd.dddEnn

The d's represent numerical characters in the data and the n's represent the 2-digit power of ten of the exponent (preceded by a sign). Either the sign, the decimal point, or the entire exponent part can be omitted. If the sign is omitted, the number is assumed to be positive; if the decimal point is omitted, it is assumed to appear after the rightmost decimal character. If the exponent is omitted, the power of ten is taken as zero.

Some examples of floating-point values are as follows:

Typed Numbers	Values Accepted
16	0.16×10^2
.16E02	0.16×10^2
1600.E-02	0.16×10^2

6.7.7 Output Formats

6.7.7.1 E and I Formats -- Integer values are always printed as the sign and a maximum number of four characters with spaces replacing

leading zeros. On output, integers are left justified within the stated field. Sufficient trailing spaces are printed to fill the field followed by one additional space.

Floating-point values are printed in a floating-point format which consists of sign, leading zero, decimal point, six decimal characters, the character E, the sign of the exponent (minus or plus), and an exponent value of two characters. For example:

Integer Values	Output Format
-1043	-1043
-0016	- 16
+0016	+ 16

Floating-point values are printed as follows:

S0.dddddd Esnn

where: S represents the sign, minus sign, or space
dddddd represents six decimal digits of the data word
E indicates exponential representation
s represents the sign of the exponent value
nn represents the exponent value

Some examples of floating-point output are:

Decimal Value	Output Format
-8,388,608.0	-0.838860E+07
-.000119209	0.119209E-03

6.7.7.2 FORMAT Control Specifications -- In most cases when data is to be presented, it must be labeled and arranged properly on a data sheet. In order that this can be accomplished with FORTRAN, a

provision has been made so that text information and spacing can be printed along with the data words. These features are provided by the special FORMAT control elements quote (") and slash (/). The slash character causes a return to the left margin.

6.7.7.3 Hollerith Output -- When text information is enclosed in quotes and is contained as part of a FORMAT statement, it is output to the specified device as it appears in the statement. This output occurs when a TYPE or WRITE statement references a FORMAT statement containing text, and all other elements of the FORMAT statement previous to the text have been used. All legal terminal characters (other than the quote character itself) can be contained within quotes and output as text.

```
        TYPE 10
10      FORMAT (/,"THIS IS HOLLERITH",/)

        TYPE 100,AMIN,AMAX
100     FORMAT (/,"MINIMUM=",E,/,"MAXIMUM=",E,/)

        TYPE 210
210     FORMAT (/,/,"    CUMULATIVE DISTRIBUTION",/,/'
        "       INCREMENTS      FREQUENCY",/)
        DO 220 K=1,100
        TYPE 250,K,VALU(L),VALU(K+1),COUNT(K)
        CONTINUE
250     FORMAT (I," ",E," ",E," ",E,/)
```

6.8 IMPLEMENTATION NOTES

6.8.1 Double Subscripts

This version of FORTRAN does not have the facility for double-subscripted variables. To accomplish double subscripting, the programmer has to include indexing statements in the source program as illustrated below. In this example, the matrices are stored columnwise in memory; that is, sequential locations in memory are used as follows:

Element	Relative Position in Memory (INDX)
a11	1
a21	2
a31	3
a41	4
a51	5
a61	6
a12	7
a22	8
.	.
.	.
.	.
a56	35
a66	36

If referencing Element a56 in the array, M=5, N=6, (I=6 for a 6 by 6 array), and INDX=M+I*(N-1)=5+I*5=35. If referencing Element a22, INDX=2+6*1=8.

```
C       MATRIX MULTIPLY PROGRAM
        DIMENSION A(36),B(36),C(36)
C       ACCEPT DIMENSION OF ARRAY
        ACCEPT 1,I
1       FORMAT (I)
        DO 10 M=1,I
        DO 10 N=1,I
        INDX=M+I*(N-1)
C       ACCEPT FIRST MATRIX
        ACCEPT 1,A(INDX)
2       FORMAT(E)
10      CONTINUE
        TYPE 15
15      FORMAT (/,/,/)
        DO 20 M=1,I
        DO 20 N=1,I
        INDX=M+I*(N-1)
C       ACCEPT SECOND MATRIX
        ACCEPT 1,B(INDX)
        C(INDX)=0
20      CONTINUE
C       MULTIPLY MATRICES
```

```
            DO 30 M=1,I
            DO 30 N=1,I
            DO 30 K=1,I
            IC=N+I*(M-1)
            IA=N+I*(K-1)
            IB=K+I*(M-1)
            C(IC)=C(IC)+A(IA)*B(IB)
30          CONTINUE
            TYPE 15
C           PRINT RESULTS IN MATRIX FORM
            DO 40 M=1,I
            TYPE 21
            DO 40 N=1,I
            INDX=M+I*(N-1)
            TYPE 1,C(INDX)
40          CONTINUE
21          FORMAT (/)
            TYPE 15
            END
```

6.8.2 Substatement Feature

The most important result of treating the equal sign as a binary operator (as explained in Section 6.4.3, Arithmetic Statements) is that it may be used more than once in arithmetic statement. In addition to simple replacement operations (see Section 6.5.3.2, Multiple Replacement), consider the following:

```
CPRM=(CKL-CKG)/(CPG=P*(Q+1.))
```

The internal arithmetic statement (or substatement), CPG=P*(Q+1), is set off from the rest of the statement by parentheses. The complete statement is a concise way of expressing the following common type of mathematical procedure:

$$\text{Let:} \quad C^1 = \frac{C_{kl} - C_{kg}}{C_{pg}}$$

$$\text{Where:} \quad C_{pg} = p(q+1)$$

The stating of a relation followed by the conditions for evaluating any of the variables can be expressed in a single arithmetic statement in FORTRAN.

A second use of the equal sign is shown below. For background on this short program, see the discussion of the same problem in the section on the IF statement.

```
C       SUM OF THE FIRST 50 INTEGERS
        KSUM=0
        INUM=50
2       KSUM=INUM+KSUM
        IF (INUM=INUM-1)3,3,2
3       STOP
```

In this example, the sum is formed by counting down, but the same results are achieved as in the section on the IF statement. The initialization is changed so that INUM starts with the value of 50 instead of 0, and the statement, INUM=INUM+1, is no longer required.

6.8.3 Error Checking

Because of the extremely compact nature of the FORTRAN-D Compiler, either FORTRAN features or error checking will suffer. In the case of FORTRAN-D, checking for certain errors is not as important as preserving the language. Therefore, the programmer is advised to follow the rules as stated in this manual and carefully check his program for mistakes. For example, the statement,

```
A = B + C -
```

will compile, although at execution time it will give unpredictable results.

It should be noted that data areas must not extend below location 5600 in FORTRAN-D. No diagnostic is issued unless program and data areas actually overlap. A maximum of 896_{10} words are available for data. Care should be taken not to exceed the limits through use of large arrays, etc. Similar obvious errors are accepted by the Compiler; their effects are often unpredictable.

6.8.4 FORTRAN-D Source Program Restrictions

The following limits are imposed upon all FORTRAN-D source programs:

1. Not more than 896 data cells. This includes all dimensional variables, user-defined variables, constants, and all constants generated by the usage of a DO loop.
2. Not more than 20 undefined forward references to unique statement numbers per program. An undefined forward reference is a reference to any statement label that has not previously occurred in the program. Multiple references to the same undefined statement numbers are considered as one reference.
3. Not more than 64 different variable names per program.
4. Not more than 128 characters per input statement.
5. Not more than 40 numbered statements per program.

6.8.4 FORTRAN-D Compiler and Operating System Core Map

The Compiler occupies the following core locations:

0003-7600	Compiler plus tables
7200-7600	Compiler tables (undefined forward reference tables, etc.)

The Operating System occupies locations:

0000-5200	Operating System without disk I/O
0000-6000	Operating System for disk I/O

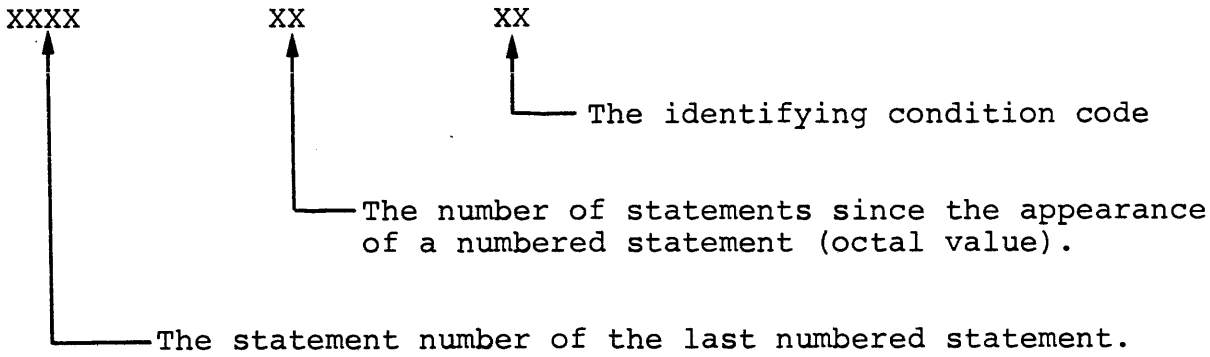
Locations 5201 through 7576 are available for the user's program when using paper tape input/output.

6.9 FORTRAN-D ERROR DIAGNOSTICS

Diagnostic procedures are provided in the Compiler to assist the programmer in program compilation. When the compiler detects errors in a FORTRAN source program, it prints the error messages on the user terminal. These messages indicate the source of the errors and direct the programmer's efforts to correct them. To speed up the Compiler process, the Compiler prints only an error code. The programmer then looks up the error message corresponding to the code in Tables 6-2 through 6-4 and takes the appropriate corrective measures.

6.9.1 Compiler Compilation Diagnostics

Format of Diagnostics



For example:

```
10      A=I(J+1)
        B=A*(H+SINF(THTA)
```

During compilation of the previous statements, the following error code would be printed:

TABLE 6-1

FORTRAN-D STATEMENT SUMMARY

Statement and Form	Explanation
Arithmetic Statements	
v = e	v is a variable (possibly subscripted); e is an expression.
Control Statements	
CONTINUE	Proceed.
DO n i=k_1,k_2,k_3	n is the statement number of a CONTINUE; i is an integer variable; $k_1k_2k_3$ are integers or nonsubscripted integer variables.
END	Terminate compilation; last statement in program.
GO TO n	n is a statement number.
GO TO (n_1,n_2,...,n_n),i	n_1,...,n_n are statement numbers; i is a nonsubscripted integer variable.
IF (e) n_1,n_2,n_3	e is an expression; n_1,n_2,n_3 are statement numbers.
PAUSE	Temporarily suspend execution.
END	Terminate compilation; last statement in program.
Specification Statements	
COMMENT	Designated by C as first character on line.
DEFINE DISK	Enables disk I/O.
DIMENSION $v_1(n_1)$, $v_2(n_2)$,... $v_n(n_n)$	v_1,...,v_n are variable names; n_1,...,n_n are integers.
FORMAT (s_1,s_2,...,s_n)	s is a data field specification.
Input/Output Statements	
ACCEPT f, list	f is a FORMAT statement number; list is a list of variables; input is from the terminal.

Table 6-1. (Cont.)

Statement and Form	Explanation
READ u,f,list	u is an integer representing the device which data is to be read: 1=terminal 2=high-speed reader, 3=disk; f is a FORMAT statement number; list is a list of variables.
TYPE f,list	f is a FORMAT statement number; list is a list of variables; output is to the terminal.
WRITE u,f,list	u is an integer representing device onto which data will be written: 1= terminal, 2=high-speed punch, 3=disk, f is a FORMAT statement number; list is a list of variables.

1Ø Ø1 11

indicating that a statement which occurs one statement octal (one decimal) after the appearance of Statement 10 is in error. The message corresponding to Code 11 shows that the number of left and right parentheses in the statement is not equal.

If a statement number is referenced but does not appear in the source program, the diagnostic code will be printed as follows:

xxxx 77 20

where the number usually reserved for the last numbered statement (xxxx) is replaced by the missing statement, e.g.

```
GO TO 1ØØ
```

The diagnostic would appear as follows where Statement 100 is never defined.

1ØØ 77 2Ø

TABLE 6-2

FORTRAN-D COMPILER COMPILATION DIAGNOSTICS

Error Code	Explanation
00	Mixed mode arithmetic expression.
01	Missing variable or constant in arithmetic expression.
03	Comma was found in arithmetic expression.
04	Too many operators in this expression.
05	Function argument is in fixed-point mode.
06	Floating-point variable used as a subscript.
07	Too many variable names in this program.
10	Program too large, core storage exceeded.
11	Unbalanced right and left parentheses.
12	Illegal character found in this statement.
13	Compiler could not identify this statement.
14	More than one statement with same statement number.
15	Subscripted variable did not appear in a DIMENSION statement.
16	Statement too long to process.
17	Floating-point operand should have been fixed-point.
20	Undefined statement number.
21	Too many numbered statements in this program.
22	Too many parentheses in this statement.
23	Too many statements have been referenced before they appear in the program.
25	DEFINE statement was preceded by some executable statement.
26	Statement does not begin with a space, tab, C, or number.

TABLE 6-3

FORTRAN-D COMPILER SYSTEMS DIAGNOSTICS

Error Code	Explanation
0240	System file error. The disk may be full.
3100	Illegal operator on compiler stack.
3417	Preprecedence error.
6204	Error in opening the compiler, FDCOMP. Must be stored under account 2.
6211	Error in reading the compiler, FDCOMP.
6463	Error in reading the compiler, FDCOMP.
6731	Disk output error.
6746	No END statement in input file.

6.9.2 Compiler Systems Diagnostics

Certain errors can make it impossible for the Compiler to proceed in the normal manner. These errors occur before the Compiler has been loaded into core. They may be caused by improperly loading the Compiler, by not having an END statement on a source file, by a machine malfunction, or by other errors. These errors, referred to as system errors, are explained in Table 6-3.

The compiler may halt (print CTRL/B followed by S) with the PC set to one of the values listed in Table 6-4 (use the WHERE command to determine the PC). The file error code will be in the AC.

TABLE 6-4
FORTRAN D ERROR HALTS

PC	Explanation
5004	Error opening FOSSIL. FOSSIL, the FORTRAN operating system, must be stored under account 2.
5011	Error reading FOSSIL.
6142	Error opening FOSL. FOSL, the FORTRAN operating system loader, must be stored under account 2.
6147	Error reading FOSL.

6.9.3 Operating System Diagnostics

Not all errors are detected by the Compiler. Some errors can only be detected by the operating system (FOSL). Also there are some conditions which indicate errors on the part of the Compiler and/or operating system. When such an error occurs during running of a program, the computer prints out an error message containing the word TILT or ERROR and an error number. The program then stops, prints BS, and returns to the Monitor.

TABLE 6-5

FORTRAN-D OPERATING SYSTEM DIAGNOSTICS

Error Code	Explanation
01	Checksum error on FORTRAN binary input.
02	Illegal origin or data address on FORTRAN binary input.
04	Disk input-output error[3].
05	High-speed reader error.
06	Illegal FORTRAN binary input device.
11	Attempt to divide by zero.
12	Floating-point input data conversion error.
13	Illegal op code.
14	Disk input-putput error[3].
15	Non-FORMAT statement used as a FORMAT.
16	Illegal FORMAT specification.
17	Floating-point number larger than 2047.
20	Square root of a negative number.
21	Exponential negative number.
22	Logarithm of a number less than or equal to zero.
40	Illegal device code used in READ or WRITE statement.

[3]May be caused by machine malfunction or operating system error.

Table 6-5. (Cont.)

Error Code	Explanation
41	System device full, could not complete a WRITE statement.
76	Stack underflow error[4].
77	Stack overflow error[4].

[4]May be caused by source program or loading error; to correct, try the following:

a. Recompile the source program.

b. Examine source program (in particular the arithmetic statements and subscripted variables).

CHAPTER 7

PAL-D ASSEMBLER

7.1 INTRODUCTION

The EduSystem 50 Assembly System is composed of the PAL-D Symbolic Assembler, LOADER, and ODTHI. The PAL-D Assembler is used to translate the user's source program into an object program (binary or machine code). LOADER is used to transfer the user's object program from the disk into core for debugging or execution. ODTHI (Octal Debugging Technique) is used to dynamically debug the object program which has been loaded into core using LOADER.

PAL-D (an acronym for Program Assembly Language for the Disk) is a two pass Assembler (with optional third pass) designed for 4K PDP-8 family of computers. A program, written in the PAL-D source language, is translated by the Assembler into a binary file in two passes through the Assembler. The binary file is loaded, by the LOADER, into the computer for execution.

During the first pass of the assembly, all user symbols are defined and placed in the Assembler's symbol table. During the second pass, the binary equivalents of the input source language are generated. The Assembler's third pass produces a printed assembly listing of the program's instructions with the location, generated binary, and source code side by side on each line. To call the PAL-D Assembler, the user types:

```
•R PALD
```

PAL-D responds by requesting INPUT: Type the name of the source program or programs to be assembled. A maximum of three files can

be assembled together. PAL-D then requests OUTPUT: Type the name of the new file in which PAL-D will store the assembled program in executable binary form. PAL-D then requests OPTION: For a normal assembly, press the RETURN key. For a listing on the line printer, respons with L. If an assembly listing is not desired, respond to OPTION with N.

PAL-D then proceeds to assemble the program: any errors in the program are indicated; the program symbol table is printed; and finally, an assembly listing of the source program is printed. When the listing is completed and the assembly finished, control is returned to the Monitor. When PAL-D begins printing the symbol table at the end of pass 2, the binary file has already been generated. Thus, the user may type CTRL/C to bypass the symbol table print out.

7.2 EduSystem 50 PAL-D

Because of the necessary hardware changes made for time-sharing on EduSystem 50, PAL-D has been revised in the following ways (as differing from PAL-D on a non-timeshared PDP-8):

a. PAL-D, under EduSystem 50, allows a very large number of user symbols in addition to the permanent symbols listed in Table 7-1. The permanent symbol table has been revised to include all instructions peculiar to the time-sharing system.

b. A CTRL/C (↑C) from the terminal terminates the assembly, and halts PAL-D, sending the user back to the Monitor.

7.3 SYNTAX

Programs processed under PAL-D are written using ASCII characters.

7.3.1 Legal Characters

The following characters are acceptable to PAL-D:

a. The alphabetic characters (ABCD...XYZ)

b. The numeric characters (0 1 2 3 4 5 6 7 8 9)

c. The special characters

␣	Space	Separates symbols and numbers.
+	Plus	Combines symbols or numbers (add)
-	Minus	Combines symbols or numbers (subtract)
!	Exclamation Mark	Combines symbols or numbers (inclusive OR)
↵	Carriage Return	Terminates a line
→|	Tabulation	Formats symbols or numbers or source tape output
,	Comma	Assigns symbolic address
=	Equal Sign	Direct assignment of symbol values
;	Semicolon	Terminates coding line (will not terminate comments)
$	Dollar Sign	Indicates end of pass
*	Asterisk	Sets current location counter; redefines origin
.	Point (Period)	Has value equal to current location counter
/	Slash	Indicates start of comment
&	Ampersand	Combines symbols or numbers (AND)
"	Quote	Generates ASCII constant
()	Parentheses	Defines literal on current page
[]	Brackets	Defines page 0 literal

d. Ignored characters

Form-Feed	Indicates the end of a logical page of source program
Blank Tape	Used for leader/trailer
Code 200	Used for leader/trailer
Rubout	Follows tabulation characters for timing purposes
Line-Feed	Follows carriage return and causes terminal paper to roll upward one line

Since certain characters are invisible (i.e., nonprinting), the following symbols are used throughout this chapter to represent their presence:

␣	Space
→|	Tabulation
↵	Carriage Return

7.3.2 Illegal Characters

All characters other than those listed above are illegal when not in a comment or TEXT field and, being illegal, their occurrence causes the error message IC (Illegal Character) to be printed by PAL-D.

7.3.3 Format Effectors

Tabulations are usually used in the body of a source program to provide a neat page; they can separate fields from one another, as between a statement and a comment. For example, a line written

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

is much easier to read if tabs are inserted to form

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

Either the ";" (semicolon) or "↵" (carriage return-line feed) character may be used as a statement terminator. The semicolon is considered identical to carriage return-line feed except that it will not terminate a comment. Example:

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

The entire expression between the "/" (slash) and ↵(carriage return) is considered a comment.

The semicolon also allows the programmer to place several lines of coding on a single line. If, for example, he wishes to write a sequence of instructions to rotate the contents of the accumulator and link six places to the right, it might look like

```
...
RTR ↵
RTR ↵
RTR ↵
...
```

The programmer may place all three RTRs on a single line by separating them with the special character ";" and terminating the line with a carriage return. The above sequence of instructions can then be written

```
RTR; RTR; RTR
```

This format is particularly useful when setting aside a section of data storage for a list. For example, a 12-word list could be reserved by specifying the following format.

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

A neat printout (or program listing) makes subsequent editing, debugging, and interpretation much easier than when the coding is laid out in a haphazard fashion.

7.4 NUMBERS

Any sequence of numbers delimited by a punctuation character is interpreted numerically by PAL-D.

The radix control pseudo-operators (pseudo-ops) indicate to the Assembler the radix to be used in number interpretation (see Section 7.9). The pseudo-op DECIMAL indicates that all numbers are to be interpreted as decimal until the next occurrence of the pseudo-op OCTAL. The pseudo-op OCTAL indicates that all numbers are to be interpreted as octal until the next occurrence of the pseudo-op DECIMAL.

The radix is initially set to octal and remains octal unless otherwise specified.

7.4.1 Arithmetic and Logical Operators

The arithmetic and logical operators are:

+	Plus	2s complement addition (modulo 4096)
-	Minus	2s complement subtraction (modulo 4096)
!	Exclamation Mark	Boolean inclusive OR (union)

&	Ampersand	Boolean AND (intersection)
␣	Space	Interpreted as inclusive OR when used to separate two symbolic operators. Example:

```
TAG,    CLA␣CLL ↵
```

7.4.2 Evaluating Expressions

Symbols and numbers (exclusive of pseudo-op symbols) may be combined by using the arithmetic and logical operators to form expressions. Expressions are evaluated from left to right. Example:

	A	B	A+B	A-B	A!B	A&B
Value	0002	0003	0005	7777	0003	0002
Value	0007	0005	0014	0002	0007	0005
Value	0700	0007	0707	0671	0707	0000

7.5 STATEMENTS

PAL-D source programs are usually prepared on a terminal with the aid of the Editor as a sequence of statements. Each statement is written on a single line and is terminated by a carriage return-line feed sequence. PAL-D statements are virtually format free; that is, elements of a statement are not placed in numbered columns with rigidly controlled spacing between elements, as in punched-card oriented assemblers.

There are four types of elements in a PAL-D statement which are identified by the order of appearance in the statement, and by the separating, or delimiting character which follows or precedes the element.

Statements are written in the general form

label, operator operand /comment

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

7.5.1 Labels

A label is the symbolic name created by the source programmer to identify the position of the statement in the program. If present, the label is written first in a statement and terminated by a comma.

7.5.2 Operators

An operator may be one of the mnemonic machine instruction codes (Tables 7-1 &7-2),or a pseudo-operation (pseudo-op) code which directs assembly processing. The assembly pseudo-op codes are described in Section 7.9. Operators are terminated with a space if an operand follows or with a semicolon, slash, or carriage return.

7.5.3 Operands

Operands are usually the symbolic address of the data to be accessed when an instruction is executed, or the input data or arguments of a pseudo-op. In each case, interpretation of operands in a statement depends on the statement operator. Operands are terminated by a semicolon, a slash if a comment follows, or a carriage return-line feed.

7.5.4 Comments

The programmer may add notes to a statement following a slash mark. Such comments do not affect assembly processing or program execution, but are useful in the program listing for later analysis or debugging.

7.6 SYMBOLS

The programmer may create symbols to use as statement labels, as operators, and as operands. A symbol is a string of one or more alphanumeric characters delimited by a punctuation character. A symbol contains from one to six characters from the set of 26 alphabetic characters and ten digits 0 through 9; however, the first character must be alphabetic.

7.6.1 Symbol Distinction

The PAL-D Assembler makes a distinction between the types of symbols it is processing. These types are

a. Permanent symbols

    JMS A symbol whose value of 4000(octal) is taken from PAL-D's permanent operation code symbol table.

b. User-defined symbols

    HERE A user-defined symbol; when used as a symbolic address tag, its value is the address of the statement it tags (this value is assigned by PAL-D).

7.6.1.1 Permanent Symbols

PAL-D has in its permanent symbol table definitions of its operation codes, operate commands, and many input-output transfer (IOT) microinstructions (see table 7-2). PAL-D's permanent symbols may be used without prior definition by the user.

7.6.1.2 User-Defined Symbols

User-defined symbols are composed according to the following rules.

a. The characters must be alphabetic (A-Z) or numeric(0-9).

b. The first character must be alphabetic.

c. Only the first six characters of any symbol are meaningful to PAL-D; the remainder, if any, are ignored.

Note that because of the third rule above, a symbol such as INTEGER would be interpreted as INTEGE since the seventh character is ignored. Remember, if symbols of more than six characters are used, the programmer must avoid defining two apparently different symbols whose first characters are identical. For example, the two symbols GEORGE1 and GEORGE2 differ only in the seventh character, thus the Assembler treats them as being the same symbol, GEORGE.

When the symbol following the space is a user-defined symbol, the space acts as an address field delimiter. Example:

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

where A is a user-defined symbol with the value 2117. The expression JMP A is evaluated as follows.

JMP	101	000	000	000	(binary representation of permanent symbol JMP)
Address A	000	011	001	111	(binary representation of address A)

The operation codes (op codes) are inclusively ORed to form

JMP A 101 011 001 111

or written more concisely in octal as 5317.

7.6.2 Symbolic Addresses

A symbol used as a label to specify a symbolic address must appear first in the statement and must be immediately followed by a comma. When used in this way, a symbol is said to be defined. A defined symbol can reference an instruction or data word at any point in the program. A symbol can be defined as a label only once. If a programmer attempts to define the same symbol as a label again, the second or successive attempt is ignored and an error is indicated. The Assembler recognizes only the first definition. These are legal symbolic addresses:

ADDR,

TOTAL,

SUM,

The following symbolic addresses are illegal:

7ABC, (first character must be alphabetic)

LAB , (comma must immediately follow label)

7.6.3 Symbolic Operators

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

TAD	(a mnemonic machine instruction operator)
PAGE	(an Assembler pseudo-op)
ZIP	(legal only if defined by the user)

7.6.4 Symbolic Operands

Symbols used as operands must have a value defined by the user. These may be symbolic references to previously defined labels where the arguments to be used by this instruction are to be found, or the values of symbolic operands may be constants or character strings.

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

The values of the two symbols AC1 and TAG, already defined by the user, are combined by a two's complement add. This value is used as the address of the operand.

7.6.5 Symbol Tables

The Assembler processes symbols in source program statements by referencing its symbol tables which contain all defined symbols along with the binary value assigned to each symbol.

Initially, the Assembler's permanent symbol table contains the mnemonic op codes of the machine instructions and the Assembler pseudo-op codes, as listed in tables 7-1 and 7-2. As the source program is processed, symbols defined in the source program are added to the user's symbol table.

7.6.6 Direct Assignment Statements

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

symbol = value

where the value may be a number or expression. For example,

```
ALPHA=5
BETA=17
```

A direct assignment statement may also be used to give a new symbol the same value as a previously defined symbol.

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

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

The value assigned to a symbol may be changed.

```
ALPHA=7
```

changes the value assigned to the first example from 5 to 7.

The user may also define symbols by use of the comma. When the first symbol of a statement is terminated by a comma, it is assigned a value equal to the current location counter (CLC). For example,

```
          *100
TAG,      CLA                  /SET CLC(ORIGIN) TO 100
          JMP A
B,        0
A,        DCA B
```

The symbol TAG is assigned a value of 0100, the symbol B a value of 0102, and the symbol A a value of 0103.

Direct assignment statements do not generate instructions or data in the object program. These statements are used to assign values so that symbols can be conveniently used in other statements.

7.7 ADDRESS ASSIGNMENTS

The PAL-D Assembler sets the origin, or starting address, of the source program to absolute location (address) 0200 unless the origin is specified by the programmer. As source statements are processed, PAL-D assigns consecutive memory addresses to the instructions and data words of the object program. This is done by incrementing the location counter each time a memory location is assigned. A statement which generates a single object program storage word increments the location counter by one. Another statement may generate six storage words, thus incrementing the location counter by six.

TABLE 7-1

EDUSYSTEM 50 SYMBOL LIST

Code	Mnemonic	Operation	Event Time
Memory Reference Instructions			
0000	AND	Logical AND	
1000	TAD	Twos complement add	
2000	ISZ	Increment and skip if zero	
3000	DCA	Deposit and Clear AC	
4000	JMS	Jump to subroutine	
5000	JMP	Jump	
6000	IOT		
7000	OPR	Operate	
Group 1 Operate Microinstructions			
7000	NOP	No operation	1
7001	IAC	Increment AC	3
7004	RAL	Rotate AC and link left one	3
7006	RTL	Rotate AC and link left two	3
7010	RAR	Rotate AC and link right one	3
7012	RTR	Rotate AC and link right two	3
7020	CML	Complement link	2
7040	CMA	Complement AC	2
7100	CLL	Clear link	1
7200	CLA	Clear AC	1

Table 7-1. (Cont.)

Code	Mnemonic	Operation	Event Time
Group 2 Operate Microinstructions			
7402	HLT	Halts the computer	4
7404	OSR	Inclusive OR switch register with AC	3
7410	SKP	Skip unconditionally	1
7420	SNL	Skip on nonzero link	1
7430	SZL	Skip on zero link	1
7440	SZA	Skip on zero AC	1
7450	SNA	Skip on nonzero AC	1
7500	SMA	Skip on minus AC	1
7510	SPA	Skip on plus AC (zero is positive)	1
Combined Operate Microinstructions			
7041	CIA	Complement and increment AC	1
7120	STL	Set link to 1	1
7204	GLK	Get link (put link in AC, bit 11)	1
7240	STA	Set AC = -1	1
7604	LAS	Load AC with switch register	1

PSEUDO-OPERATORS

DECIMAL	OCTAL
EXPUNGE	PAGE
FIELD	PAUSE
FIXTAB	TEXT
I	XLIST
	Z

Direct assignment statements and some Assembler pseudo-ops do not generate storage words and therefore do not affect the location counter.

7.7.1 Current Address Indicator

The special character . (point or period) always has a value equal to the value of the current location counter. It may be used as any integer or symbol (except to the left of an equal sign). Example:

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

is equivalent to JMP 0202. Also,

```
*300
.+2400
```

will produce in location 0300 the quantity 2700. Consider

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

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

7.7.2 Indirect Addressing

When the character appears in a statement between a memory reference instruction and an operand, the operand becomes the address containing the address of the statement to be executed. Consider

```
TAD 40
```

which is a direct address statement, where 40 is interpreted as the address containing the quantity to be added to the accumulator. Thus, if address 40 contains 0432, then 0432 is added to the accumulator. Now consider

```
TAD I 40
```

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

Then a reference is made to an address not on the same page as the reference, PAL-D sets the indirect bit (bit 3) of the machine instruction, generating an indirect address linkage to the off-page reference (see Paging and Off-Page Referencing, Sections 7.8.1.1 and 7.8.1.2).

In the case of several off-page references to the same address, the indirect address linkage will be generated only once. Example:

```
          *2117
A,        CLA
           .
           .
           .
          *2600
          TAD A
           .
           .
           .
          DCA A
```

The space preceding the user-defined symbol A acts as an address field delimiter. PAL-D will recognize that the address tag A is not on the current page (in this case 2600-2777) and will generate a link to it in the following manner. In location 2600, PAL-D will place the word

1777 (octal equivalent of TAD I 2777)

and in location 2777 (the last location on the current page) the word 2117 (the actual address of A) will be placed. When it sees the second reference to A it will use the previous link word rather than creating a new one.

PAL-D will recognize and generate an indirect address linkage only when the address referenced is to a location on another page, not the current page. The programmer must use the character I to indicate an explicit indirect address when indirectly addressing to a location on the current page.

PAL-D cannot generate a link for an instruction that is already specified as being an indirect address. In this case, PAL-D will type the error message II (Illegal Indirect); the error message is ignored and assembly is continued.

7.7.3 Autoindexing

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

It must be remembered that initially these locations (10 to 17) must be set to one less than the first desired address. Because of their characteristics, these locations are called autoindex registers. No incrementation takes place when locations 10 to 17 are addressed directly. Example:

```
        Statement is in location 500
        Data is on the page starting at 5000
        Autoindexing register 10 is used for addressing

0476  1377  TAD (5000-1)        /SET UP AUTO INDEX
0477  3010  DCA 10              /WITH 4777
0500  1410  TAD I 10           /C(10) IS INCREMENTED TO 5000 BEFORE
  .     .                      /IT IS USED AS AN ADDRESS
  .     .
  .     .
0577  4777                     /LITERAL GENERATED BY PAL-D
```

When the statement in location 500 is executed, the content of location 10 will be incremented to 5000 and the content of location 5000 will be added to the content of the accumulator. If the instruction TAD I 10 is re-executed, the content of location 5001 is added to the content of the accumulator, and so on.

7.7.4 Literals

Symbolic and integer literals (constants) may be defined as shown below:

```
CLA
TAD (2)
DCA INDEX
```

Operator and operand must always be separated with a space.

The left parenthesis is a signal to the Assembler that the integer following is to be assigned a location in the table at the top of the current page. This is the same table in which the indirect address linkages are stored. In the above example, the quantity 2 is stored in the first free location in a list beginning at the top of the current page (relative address 177), and the statement in which it appears is encoded with an address referring to that location.

A literal is assigned to storage the first time it is encountered; subsequent references will be to the same location.

If the programmer wishes to assign literals to page 0 rather than the current page, he must use square brackets, [ ], in place of paren-

theses. Whether using parentheses or square brackets, the right or closing member is optional and may always be replaced with a carriage return.

```
TAD (777
```

Nesting - Literals may be nested as shown below.

```
*200
TAD(TAD(30
```

will generate

```
0200  1276
  .     .
  .     .
  .     .
0376  1377
0377  0030
```

(literals assigned to locations 0377 and 0376; top of current page)

This type of nesting may be carried to many levels.

Literals are stored on each page starting at relative address 177 (only 127_{10} or 177_8 literals may be placed on page 0). If literals are being generated for some nonzero page and then the origin is set to another page, the current page literal buffer is punched out during pass 2. If the origin is reset to the previously used page, the same literal will be generated if used again.

If a single character is preceded by a quote ("), the 8-bit value of the ASCII code for that character is inserted instead of taking the letter as a symbol.

Example:

```
CLA
TAD ("A

...
```

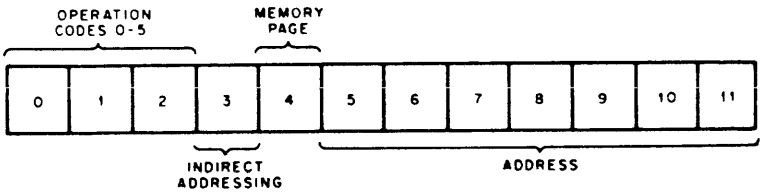
will place the constant 0301 in the accumulator.

7.8 INSTRUCTIONS

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

7.8.1 Memory Reference Instructions

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



Memory Reference Instruction Bit Assignments

Bits 0 through 2 contain the operation code of the instruction to be performed (such as AND,TAD, or JMP). Bit 3 tells the computer if the instruction is indirect, that is, if the address of the instruction specifies the location of the operand, or if it specifies the location of the address of the operand. Bit 4 tells the computer if the instruction is referencing the current page or page zero. This leaves bits 5 through 11 (7 bits) to specify an address. In these 7 bits, 200

octal or 128 decimal locations may be specified; the page bit increases accessible locations to 400 octal or 256 decimal.

The address field of a memory reference instruction may be any valid expression. Example:

```
A=270
*200
TAD A-20
```

produces, in location 200, the word

1250

which in binary is 001 010 101 0000

which is also TAD 250.

7.8.1.1 Paging

To ease the programmer's addressing problems, a convention has been defined that devides memory into sectors called pages. Each page contains 200 octal locations (128 decimal) numbered 0 to 177 (octal) on that page. There are 40 octal or 32 decimal pages numbered 0 to 37 (octal). Some examples of page numbers and the absolute and relative locations (addresses) are shown below. It must be borne in mind, however, that there is no physical separation of pages in memory.

Page	Absolute Address	Relative Address
0	0 - 177	0 - 177
1	200 - 377	0 - 177
2	400 - 577	0 - 177
36	7400 - 7577	0 - 177
37	7600 - 7777	0 - 177

The following table offers a comparison of specific absolute and relative addresses on the same page.

Page	Absolute Address	Relative Address
0	10	10
3	617	17
12	2577	177
31	6255	55
37	7777	177

Since only seven bits are necessary to address 200 octal locations, bits 5 to 11 are reserved for this function.

7.8.1.2 Off-Page Referencing

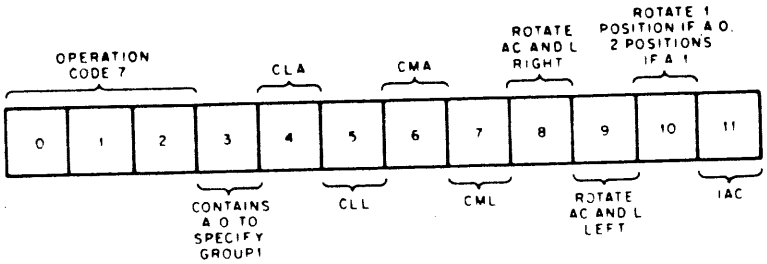
The page on which an absolute address is contained can be determined from bit 4 of the instruction. If bit 4 is a 0, the address refers to a location on page 0; if bit 4 is a 1, the address refers to a location on the current (same) page, that is, the same memory page as the instruction.

7.8.2 Augmented Instructions

Augmented instructions are divided into two groups: operate and input-output transfer microinstructions.

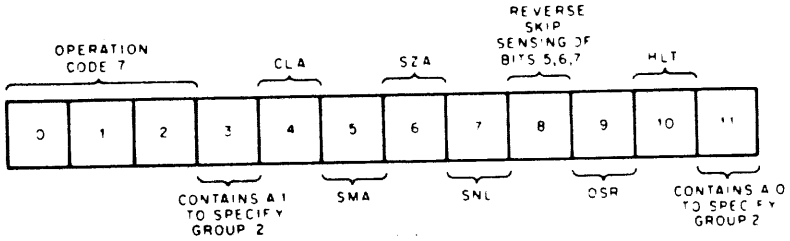
7.8.2.1 Operate Microinstructions

Within the operate group there are two groups of microinstructions. Group 1 microinstructions are principally for clear, complement, rotate, and increment operations and are designated by the presence of a 0 in bit 3 of the machine instruction word. (See table 7-1.)



Group 1 Operate Microinstruction Bit Assignments

Group 2 microinstructions are used principally in checking the content of the accumulator and link and, based on the check, continuing to or skipping the next statement. Group 2 microinstructions are identified by the presence of a 1 in bit 3 and a 0 in bit 11 of the machine instruction word. (See table 7-1.)



Group 2 Operate Microinstruction Bit Assignments

Group 1 and group 2 microinstructions can not be combined because bit 3 determines only one or the other.

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

OR Group	AND Group
SMA	SPA
SZA	SNA
SNL	SZL

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

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

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

SZA SNL

The next statement is skipped if

the accumulator contains 0000, or
the link is a 1, or
both conditions exist.

b. AND Group - If the skips are combined in a statement, the logical AND of the conditions determines the skip.

SNA SZL

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

7.8.2.2 Input-Output Transfer Microinstructions

These microinstructions initiate operation of peripheral equipment and effect information transfer between the central processor and the input-output device (s). This is the principal function of the input-output transfer (IOT) microinstructions. Table 7-2 lists all valid IOT microinstructions, and each is discussed in detail in Chapter 11.

TABLE 7-2

EDUSYSTEM 50 IOT INSTRUCTION SUMMARY

Number	Instruction	Function
Program Control		
6200	CKS	Check Status
6402	DUP	Duplex Console
6403	UND	Unduplex Console
6405	CLS	Clear Status
6411	URT	User Run Time
6412	TOD	Time of Day
6413	RCR	Return Clock Rate
6414	DATE	Date
6415	SYN	Quantum Synchronization
6416	STM	Set Timer
6417	SRA	Set Restart Address
6420	TSS	Skip on TSS/8
6421	USE	User
6422	CON	Console
6430	SSW	Set Swtich Register
6431	SEA	Set Error Address
6440	ASD	Assign Device
6442	REL	Release Device
7402	HLT	Halt
7404	OSR	OR With Switch Register
File Control		
6406	SEGS	Segment Count
6600	REN	Rename File
6601	OPEN	Open File
6602	CLOS	Close File
6603	RFILE	Read File

Table 7-2. (Cont.)

Number	Instruction	Function
6604	PROT	Protect File
6605	WFILE	Write File
6610	CRF	Create File
6611	EXT	Extend File
6612	RED	Reduce File
6613	FINF	File Information
6614	SIZE	Segment Size
6616	WHO	Who
6617	ACT	Account Number
Input Buffer Control		
6030	KSR	Read Keyboard String
6031	KSF	Skip on Keyboard Flag
6032	KCC	Clear Keyboard Flag
6034	KRS	Read Keyboard Buffer Static
6036	KRB	Read Keyboard Buffer Dynamic
6400	KSB	Set Keyboard Break
6401	SBC	Set Buffer Control Flags
Output Buffer Control		
6040	SAS	Send A String
6041	TSF	Skip On Teleprinter Flag
6042	TCF	Clear Teleprinter Flag
6044	TPC	Load Teleprinter and Print
6046	TLS	Load Teleprinter Sequence
High-Speed Paper Tape Reader and Control		
6010	RRS	Read Reader String
6011	RSF	Skip On Reader Flag
6012	RRB	Read Reader Buffer
6014	RFC	Reader Fetch Character

Table 7-2. (Cont.)

Number	Instruction	Function
High-Speed Paper Tape Punch and Control		
6020	PST	Punch String
6021	PSF	Skip On Punch Flag
6022	PCF	Clear Punch Flag
6024	PPC	Load Punch Buffer and Punch Character
6026	PLS	Load Punch Buffer Sequence
DECtape Control		
6764	DTXA	Load Status Register A
6771	DTSF	Skip On Flags
6772	DTRB	Read Status Register B
Line Printer		
6660	LST	Print String
6662	LCF	Clear Printer Flag
6661	LSF	Skip on Printer Flag
6664	LLC	Print Character
6666	LPC	Print Character
Card Reader		
6632	RCRA	Read Card, Alpha
6634	RCRB	Read Card, Binary
6636	RCRC	Read Card, Compressed
Disk Cartridge		
6743	DLAG	Perform Disk Transfer
6772	RDS	Read Device Status

7.9 PSEUDO-OPERATORS

The programmer may use pseudo-operators (pseudo-ops) to direct the Assembler to perform certain tasks or to interpret subsequent coding in a certain manner. Some pseudo-ops generate storage words in the object program, other pseudo-ops direct the Assembler on how to proceed with the assembly. Pseudo-ops are maintained in the Assembler's permanent symbol table.

The function of each PAL-D pseudo-op is described below.

7.9.1 Current Location Counter

The programmer may use the PAGE pseudo-op to reset the current location counter (CLC) to the first location on a specified page.

PAGE — without an argument, the CLC is reset to the first location on the next succeeding page. Thus, if a program is being assembled into page 1 and the programmer wishes to begin the next segment of his program on page 2, he need only insert PAGE, as follows.

```
JMP .-7        (Last location used on page 1)
PAGE
CLA            (First location on page 2)
```

PAGE n — resets the CLC to the first location of page n, where n is an integer, a previously defined symbol, or a symbolic expression. Example:

```
                (sets the CLC to location 400)
PAGE 2          (sets the CLC to location 1400)
PAGE 6
```

7.9.2 Extended Memory

When using more than one memory bank, the pseudo-op FIELD instructs the Assembler to output a field setting.

FIELD n — where n is an integer, a previously defined symbol, or a symbolic expression within the range of $0 \leq n \leq 7$.

This pseudo-op causes a field setting (binary word) of the form

11 XXX 000 where $000 \leq XXX \leq 111$

to be output on the binary file during pass 2. This word is interpreted by the Loader, which then begins loading information from the file into the new field. (Field settings are ignored by TSS/8 LOADER.)

7.9.3 RADIX Control

Integers used in a source program are usually taken as octal numbers. If, however, the programmer wishes to have certain numbers treated as decimal, he may use the pseudo-op DECIMAL.

DECIMAL	all integers in subsequent coding are taken as decimal until the occurrence of the pseudo-op OCTAL.
OCTAL	resets the radix to its original octal base.

7.9.4 Listing Control

During pass 3, a listing of the source program is printed. The programmer may, however, control the output of his pass 3 listing by use of the pseudo-op XLIST.

XLIST	Those portions of the source program enclosed by XLIST will not appear in the pass 3 listing.

7.9.5 Text Facility

The pseudo-op TEXT enables the user to represent a character or string of characters in ASCII code trimmed to six bits and packed two characters to a word. The numerical values generated by TEXT are left-justified in the storage words they occupy, with the unused bits of the last word filled with 0s.

A string of text may be entered by giving the pseudo-op TEXT followed by a space, a delimiting character, a string of text, and the same delimiting character. Example:

```
TEXT ATEXT STRINGA
```

The first printing character following TEXT is taken as the delimiting character, and the text string is the characters which follow until the delimiting character is again encountered.

If the example above were at location 0200, the pass 3 listing would be as follows.

```
0200  2405          TEXT ATE
0201  3024  XT
0202  4023   S                    (␣ denotes a space)
0203  2422  TR
0204  1116  IN
0205  0700  GA
```

NOTE

With TEXT, any printing character may be used as a delimiting character; the delimiting character cannot be used in the text string.

7.9.6 End of Program

The special symbol $ (dollar sign) indicates the end of a program. When the Assembler encounters the $, it terminates the pass.

7.9.7 End of File

The pseudo-op PAUSE signals some assemblers to stop processing the current input file. TSS/8 PAL-D ignores any PAUSE statements.

7.9.8 Altering the Symbol Table

PAL-D has a permanent symbol table which contains all instructions (symbols and their octal values) required by EduSystem 50. They are referred to as PAL-D's basic instructions or symbols, and are listed in tables 7-1 and 7-2.

When the symbolic program to be assembled required instructions not already in the table (e.g., card reader IOT's), the table must be altered to include those instructions. PAL-D has two pseudo-ops that are used to alter the permanent symbol table:

EXPUNGE	deletes the entire permanent symbol table, except pseudo-ops.
FIXTAB	appends symbols to the table for duration of the assembly. All symbols defined before the occurrence of FIXTAB are temporarily made part of the permanent symbol table.

These pseudo-ops can be used to eliminate unneeded symbols from the table, thus providing more storage for user symbols.

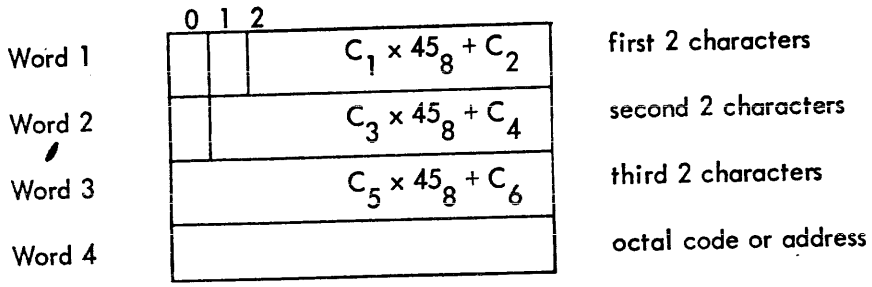
To append the following card reader IOT's to the symbol table, the programmer generates an ASCII file containing:

```
RCSF=6631
RCSP=6671
RCRD=6674
FIXTAB
```

This file is then included as one of the input files to PAL-D. (Only the last input file should contain a dollar sign.)

7.9.9 Internal Representation

Each permanent and user-defined symbol occupies four words (locations) in the symbol table storage area, as shown below:



where C_1, C_2,...C_6 represent the first character, second character,..., sixth character resprectively. (Symbols may consist of from one to six characters.) Bits 0 and 1 of word 1 and bit 0 of word 2 are system flags. With a permanent symbol, word 4 contains the octal code of the symbol; with a user-defined symbol, word 4 contains the address of the symbol. For example: the permanent symbol TAD is represented as follows.

Word 1 = $24_8 \times 45_8 + 01 = 1345_8$ or TA

Word 2 = $04_8 \times 45_8 + 00 = 224_8 + \underbrace{4000}_{\text{flag bit}} = 4224_8$ D

Word 3 = 0000

Word 4 = 1000 (octal code for TAD)

Note that the first degit of the ASCII octal code for each character is always trimmed by the assembler so that the character is represented using six bits of a word. For example, ASCII code for T is 324, it was trimmed to 24; A is 301, it was trimmed to 01; etc.

7.10 PROGRAM PREPARATION AND ASSEMBLER OUTPUT

The source language file is prepared using the Editor.

7.10.1 Program File

Since the Assembler ignores certain characters, these may be used

freely to produce a more readable symbolic source file. These useful characters are tab and form-feed.

The Assembler will also ignore extraneous spaces, carriage return-line feed combinations, and rubouts.

The program body consists of statements and pseudo-ops. The program is terminated by the dollar sign ($). If the program is large, it may be split into as many as three files. This often facilitates editing the source program since each section is physically smaller.

The Assembler initially sets the origin (current location counter) of the source program to 0200. The programmer may reset the current location counter by use of the asterisk.

The following two programs are identical except that format effectors were used in the second printout.

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

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

Both of these programs will produce the same binary code. The second, however, is easier to read.

7.10.2 Assembly

PAL-D is a two-pass assembler with an optional third pass which produces a side-by-side assembly listing of the symbolic source statements, their octal equivalents, and assigned absolute addresses. These passes are invisible to the user. However, the user determines whether or not the third pass will be made by his response to PAL-D's OPTION: query (see Section 7-11).

7.10.3 Pass 1

During pass 1, PAL-D processes the source (file) and places in its user's symbol table the definitions of all symbols used. The

user's symbol table is printed at the end of pass 2. If any symbols remain undefined at the end of pass 1, the US (Undefined Symbol) diagnostic is printed during pass 2 when the undefined symbol is encountered (see Error Diagnostics). The symbol table is printed in alphabetical or er on the terminal. If the program listed above were assembled, PAL-D would output the following symbol table.

```
BEGIN  0200        CHAR   0213        MSPACE 0214
```

7.10.4 Pass 2

During pass 2, PAL-D processes the source or file and generates binary output using the symbol table equivalences defined during pass 1. The binary output may be loaded in core by LOADER.

The binary coded file consists of leader code, an origin setting, and data words. Every occurrence in the source program of an asterisk causes a new origin setting in the binary output. At the end of the binary coded file, a binary checksum is produced and trailer code is generated.

7.10.5 Pass 3

During pass 3, PAL-D processes the source file and prints out a side-by-side listing of the generated octal code and the original source language. If the program shown above were assembled, the pass 3 listing would be:

```
                *200
                /EXAMPLE OF FORMAT
0200    0000    BEGIN,  0                       /START OF PROGRAM
0201    6032            KCC
0202    6031            KSF                     /WAIT FOR FLAG
0203    5202            JMP     .-1             /FLAG NOT SET YET
0204    6036            KRB                     /READ IN CHARACTER
0205    3213            DCA     CHAR
0206    1213            TAD     CHAR
0207    1214            TAD     MSPACE          /IS IT A SPACE?
0210    7650            SNA CLA
0211    7402            HLT                     /YES
0212    5202            JMP     BEGIN+2         /NO: INPUT AGAIN
0213    0000    CHAR,   0                       /TEMPORARY STORAGE
0214    7540    MSPACE, -240                    /-ASCII EQUIVALENT
                /END OF EXAMPLE
↑BS
```

7.11 OPERATING THE PALD ASSEMBLER

Assembling with PAL-D in TSS/8 requires no operator intervention between passes. The symbol table is typed out at the end of pass two and the listing at the end of pass three. The assembly may be terminated at any point by typing CTRL/C. Control will revert from PAL-D to the Monitor program which will type out a dot (•) and wait for the next instruction from the terminal. In the illustrations which follow, non-underlined characters are those typed out by the system; underlined characters represent user-supplied data. Time sharing assemblies are requested as follows.

In response to the monitor's dot

•

the user types the command, a space and the name of the system program.

```
•R PALD
```

PAL-D is brought into core and signals its readiness by requesting an input file name.

```
INPUT:TYPE
```

The user reply in this case was BIN2, a user symbol for a source program to be assembled. PAL-D next requests the name of an output file.

```
OUTPUT:TYPEB
```

The user response was TYPE2, the name under which the assembled program will be stored. Optionally, the user may type the RETURN key to specify no output file.

```
OUTPUT:
```

This is useful in debugging. A program may be corrected and reassembled any number of times with production of an output file postponed until a satisfactory version is achieved. PAL-D's final query is whether the user wants a program listing.

```
OPTION:
```

There are three effective responses only: N signifying No, L signifying a listing on the Line printer, and ↵ (RETURN key) signifying a listing on the terminal. When it receives the final response, PAL-D reads in the user source program from disk (source programs are stored prior to assembly) and proceeds with the assembly. After assembly, PAL-D returns control to the Monitor which types

```
↑BS
.
```

and waits for the user to supply the next command.

NOTE

Under the TSS/8 Monitor PAL-D does not require a dollar sign ($) as the last entry in a source program, but if it does not find one it types a message to warn the user that his program may not be assembled properly by an assembly program other than time-sharing PAL-D.

The following listing was reproduced from a time sharing run. It illustrates the initial dialogue, the symbol table produced at the end of pass 2 (any error messages would also appear at this point) and the listing, in octal notation, produced during pass 3.

INITIAL DIALOGUE

```
.R PALD

 INPUT:R IN2
OUTPUT:TYP2
OPTION:
```

SYMBOL TABLE

```
COUNT   0415
CRLF    0417
LOOP    0406
OUT     0425
REG     0416
START   0400
```

PROGRAM LISTING

```
                        /PROGRAM TO TYPE OUT "123456789"
                        *0400
0400   7200  START,     CLA
0401   4217             JMS CRLF
0402   1377             TAD (-12
0403   3215             DCA COUNT
0404   1376             TAD (260          /ASCII FOR ZERO
0405   3216             DCA REG
0406   1216  LOOP,      TAD REG
0407   4225             JMS OUT
0410   2216             ISZ REG
0411   2215             ISZ COUNT
0412   5206             JMP LOOP
0413   4217             JMS CRLF
0414   7402             HLT
0415   0000  COUNT,     0
0416   0000  REG,       0
0417   0000  CRLF,      0
0420   1375             TAD (215          /ASCII FOR CARRIAGE RETURN
0421   4225             JMS OUT
0422   1374             TAD (212          /ASCII FOR LINE FEED
0423   4225             JMS OUT
0424   5617             JMP I CRLF
0425   0000  OUT,       0
0426   6046             TLS
0427   6041             TSF
0430   5227             JMP .-1
0431   7200             CLA
0432   5625             JMP I OUT
```

LITERALS

```
0574   0212
0575   0215
0576   0260
0577   7766
```

```
$BS
```

7.12 ERROR DIAGNOSTICS

PAL-D makes many error checks as it processes source language statements. When an error is detected, the Assembler prints an error message. 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.

PAL-D's error messages are listed and explained below.

Error Code	Explanation
BE	Two PAL-D internal tables have overlapped - This situation can usually be corrected by decreasing the level of literal nesting or number of current page literals used prior to this point on the page.
DE	Systems device error - An error was detected when trying to read or write the system device; after three failures, control is returned to the Monitor.
DF	Systems device full - The capacity of the systems device has been exceeded; assembly is terminated and control is returned to the Monitor.
IC	Illegal character - An illegal character was encountered in other than 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 other means than the equal sign; the symbol was not redefined.

IE Illegal equals - An equal sign was used in the wrong context. Examples:

```
TAD A +=B  (the expression to the left of the equal
           sign is not a single symbol or, the
           expression to the right of the equal
A+B=C      sign was not previously defined)
```

II Illegal indirect - An off-page reference was made; a link could not be generated because the indirect bit was already set. Example:

```
*200
TAD I A ↵
   .
   .
   .
PAGE↵
A,  7240 ↵
```

ND The program terminator, $, is missing.

PE Current nonzero page exceeded - An attempt was made to
a. override a literal with an instruction, or
b. override an instruction with a literal; this can be corrected by
    (1) decreasing the number of literals on the page or
    (2) decreasing the number of instructions on the page.

SE Symbol table exceeded - Assembly is terminated and control is returned to the Monitor.

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 8

UTILITY PROGRAMS

8.1 SYMBOLIC EDITOR

The EduSystem 50 Symbolic Editor (EDIT) provides the user with a powerful tool for creating and modifying source files online. Its precise capabilities and commands are detailed in Introduction to Programming, Chapter 5. EDIT allows the user to delete, insert, change, and append lines of text, and then obtain a clean listing of the updated file. EDIT also contains commands for searching the file for a given character.

EDIT considers a file to be divided into logical units, called pages. A page of text is generally 50-60 lines long, and hence corresponds to a physical page of program listing. A FORTRAN-D program is generally 1-3 pages in length; a program prepared for PAL-D may be several pages in length. EDIT operates on one page of text at a time, allowing the user to relate his editing to the physical pages of his listing. EDIT reads a page of text from the input file into its internal buffer where the page becomes available for editing. When a page has been completely updated, it is written onto the output file and the next page of the input file is made available. EDIT provides several powerful commands for paging through the source file quickly and conveniently.

NOTE

The end of a page of text is marked by a form feed (CTRL/L) character. Form feed is ignored by all EduSystem 50 language processors.

To call the Editor, type:

```
•R EDIT
```

EDIT responds by requesting INPUT: Type and enter the name of the source file to be edited. If a new file is to be created using EDIT, there is no input file. In this case, strike the RETURN key. EDIT then requests OUTPUT: Type the name of the new, edited, file to be created. The name of the output file must be different from the name of the input file. If EDIT is being called to list the input file, there is no need to create an output file; strike the RETURN key. When EDIT sets up its internal files and is ready for a command, it rings the bell on the terminal.

For example:

```
•R EDIT
 INPUT:WXZOLD
OUTPUT:XYZNEW                    (Bell rings at this point.)
```

8.2 LOADER

LOADER is used to load programs in BIN format from a disk file into the user's core area for execution. These files in BIN format can be created by PAL-D in the course of an assembly or they can be loaded from paper tape using PIP or PUTR (see the PIP section for special instructions on loading BIN format tapes).

To call LOADER, type:

```
•R LOADER
```

LOADER responds by asking for INPUT: Respond by entering the name of the file or files to be loaded. Although many System Library Programs allow multiple input files, the LOADER uses this feature to

TABLE 8-1

SYMBOLIC EDITOR OPERATIONS SUMMARY

Special Characters	Function
Carriage Return (RETURN KEY)	Text Mode - Enter the line in the text buffer.
CTRL/C	Text Mode - Same as form feed.
CTRL/U	Text Mode - Cancel the entire line of text, continue typing on next line. Command Mode - Cancel command. Editor issues a ? and carriage return/line feed.
Equal Sign (=)	Command Mode - Used in conjunction with . and / to obtain their value (for example, type .=).
Form Feed (CTRL/L Combination)	Text Mode - End of inputs, return to command mode.
Left Angle Bracket (<)	Command Mode - List the previous line (equivalent to .-1L).
Line Feed (↓)	Text Mode - Used in SEARCH command to insert a CR/LF combination into the line being searched.
Right Angle Bracket (>)	Command Mode - List the next line (equivalent to .+1L).
Rubout	Text Mode - Delete from right to left one character for each rubout typed. Does not delete past the beginning of the line. Is not in effect during a READ command. Command Mode - Same as CTRL/U.
Slash (/)	Command Mode - Value equal to number of last line in buffer. Used as argument (as in /-5,/L).
Tabulation (CTRL/TAB Key Combination)	Text Mode - Produces a tabulation which, on output, is interpreted as spaces.

TABLE 8-2

EDIT COMMAND SUMMARY

Command	Format	Meaning
APPEND	A	Append incoming text from keyboard to any already in the buffer until a form feed is encountered.
↑C	CTRL/C	Stop listing and return to Command Mode.
CHANGE	nC	Delete line n, replace it with any number of lines from the keyboard until a form feed is entered.
	m,nC	Delete lines m through n, replace from keyboard as above until form feed is entered.
DELETE	nD	Delete line n of the text.
	m,nD	Delete lines m through n inclusive.
END	E	Output the contents of the buffer. Read any pages remaining in the input file, outputting them to the output file. When everything in the input file has been moved to the output file, close it out and return to the Monitor. E is equivalent to a sufficient number of N's followed by a T command.
GET	G	Get and list the next line beginning with a tag.
INSERT	I	Insert before line 1 all the text from the keyboard until a form feed is entered.
	nI	Insert before line n until a form feed is entered.
KILL	K	Kill the buffer (i.e., delete all text lines).
LIST	L	List the entire buffer.
	nL	List line n.
	m,nL	List lines through n inclusive.
MOVE	m,n$kM	Move lines m through n inclusive to before line k.
NEXT	N	Output the entire buffer and a form feed, kill the buffer and read the next page.

Table 8-2. (Cont.)

Command	Format	Meaning
	nN	Repeat the above sequence n times.
PROCEED	P	Output the contents of the buffer to the output file, followed by a form feed.
	nP	Output line n, followed by a form feed.
	m,nP	Output lines m through n inclusive followed by a form feed.
READ	R	Read text from the input file and append to buffer until a form feed is encountered.
SEARCH	S	Search the entire buffer for the character specified (but not echoed) after the carriage return. Allow modification when found. Editor outputs a slash (/) before beginning a SEARCH.
	nS	Search line n, as above, allow modification.
	m,nS	Search lines m through n inclusive, allow modification.
TERMINATE	T	Close out the output file and return to the Monitor.

special advantage. Because it loads the files in the order they are typed, LOADER can be used to load patches and overlays. After it has requested INPUT, LOADER requests OPTION: For normal operation strike the RETURN key; LOADER is able to load any part of core, except locations 7767 - 7777. If the program to be loaded is to be debugged, respond to OPTION: with D. This will cause ODTHI to be loaded along with the input files and started. ODTHI indicates that it is ready by printing a second line feed. ODTHI uses locations 4 and 7000 - 7777; and if loaded along with a program which uses any of these locations, the result is unpredicatble.

Example 1: Normal Operation

```
•R LOADER
 INPUT:MAIN, PATCH1, PATCH2
OPTION:
↑BS
```

Example 2: Load ODT with Input File

```
•R LOADER
 INPUT:PROG1
OPTION:D
```

As seen in the first example, LOADER returns control to Monitor when it is finished. The user can then start the program by using the Monitor command START. For example, LOADER can be used to load and run the short program given as an example in the section on PAL-D.

```
•R LOADER

 INPUT:BIN2
OPTION:
↑BS
•START 400

0123456789
↑BS
```

NOTE

All BIN format files loaded by LOADER include a checksum. If LOADER detects a checksum error while loading, it prints LOAD ERROR and terminates the load.

8.3 OCTAL DEBUGGING TECHNIQUE (ODTHI)

ODTHI is a powerful octal debugging tool for testing and modifying PDP-8 programs in actual machine language. It allows the user to control the execution of his program and, where necessary, make immediate corrections to the program without the need to reassemble.

The complete command repertoire of ODT is documented in Introduction to Programming, Chapter 5. ODTHI (on EduSystem 50) is the high-core version which resides in locations 7000 through 7777. The paper-tape output commands of regular ODT are not available in EduSystem 50 ODT. To call ODTHI, the user types:

```
•LOAD 2 ODTHI 0 7000
•START 7000
```

If ODTHI is to be used to debug a program being loaded with LOADER, ODTHI can be loaded and started directly by specifying the Debug (D) option to LOADER.

ODTHI executes an SRA (Set Restart Address) as part of its initialization process. As a result, typing CTRL/C always returns control to ODTHI. If the program being debugged sets up its own restart address, typing CTRL/C transfers control to the new restart address. It is necessary to type ↑BS followed by START 7000 to force control back to ODTHI. Every time ODTHI regains control, it puts the terminal in duplex mode. Users debugging programs which do not operate in duplex mode, should be aware of this fact.

ODTHI saves the state of the delimiter mask, when it regains control via a breakpoint. The state of this mask is restored on a Continue (C) command, but not on a GO (G) command.

8.4 CATALOG (CAT)

The Monitor maintains a library of disk files for each user. The System Library Program CAT is used to obtain a catalog of the contents of this library. For each file, CAT prints the size of the file in units of disk segments. The size of a disk segment is 256 (decimal) words of disk storage. The protection code for the file is also given. (See the section on Advanced Monitor Commands for a

TABLE 8-3

ODT COMMAND SUMMARY

Command	Meaning
A	Open for modification, the register in which the contents of AC were wtored when the breakpoint was encountered.
B	Remove the breakpoint.
nnnnB	Establish a breakpoint at location nnnn.
Back Arrow (←) (SHIFT/O)	Close register, open indirectly.
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.
nnnnG	Transfer program control to location nnnn.
Illegal Character	Current line typed by user is ignored, ODT types ?CR/LF.
LINE FEED	Close register and open the next sequential one for modification.
M	Open the search mask register, initially set to 7777. It may be changed by opening the search mask register and typing the desired value after the value typed by ODT, then closing the register.
LINE FEED	Close search mask register and open next register immediately following, containing the location at which the search begins. It may be changed by typing the lower limit after the one typed by ODT, then closing the register.
LINE FEED	Close lower search register, open next register containing the upper search limit initially set to 7000 (location of ODT). It may be changed by typing the desired upper limit after the one typed by ODT and closing the register with a carriage return.
RETURN	Close previously opened register.
/	Reopen latest opened register.

Table 8-3. (Cont.)

Command	Meaning
nnnn/	Open register designated by the octal number nnnn.
Up Arrow (↑) (SHIFT/N)	Close register, take contents of that register as a memory reference and open it.
nnnnW	Search the portion of core as defined by the upper and lower limits for the octal value nnnn.

precise explanation of protection codes.) If the program was created by any of the System Library Programs, it has a protection code of 12, meaning that other users can read the file, but only the owner can change it. To call CAT, type:

```
•R CAT
```

The CAT program then prints a listing similar to the one shown below and concludes by printing ↑BS and exiting to the Monitor.

```
•R CAT

DISK FILES FOR USER  3,13 ON  9-JUN-70

NAME       SIZE  PROT    DATE
FIE   •BIN    1    17   3-JUN-70
PROG  •FCL    2    12   9-JUN-70
INTER •BAS    1    17   9-JUN-70
BAS000•TMP    1    17   9-JUN-70
BAS100•TMP    1    17   9-JUN-70
INT2  •BAC    1    37   9-JUN-70
FCLPRG•FCL    2    12   9-JUN-70

TOTAL DISK SEGMENTS:  9

↑BS
```

8.5 SYSTEM STATUS (SYSTAT)

It is frequently useful to know the status of the system as a whole; how many users are on-line, where they are, what they are

doing, etc. The SYSTAT program provides this capability. To call SYSTAT, type:

```
•SYSTAT
```

or, to produce a listing on the line printer,

```
•SYSTAT:L
```

SYSTAT responds by printing on the first line: the version of the Monitor being run, the time, and the date. SYSTAT then reports the uptime which is the length of time in hours, minutes, and seconds since the system was last put on-line.

SYSTAT then lists all on-line users. Each user is identified by his account number. The job number assigned to him and the number of the console he is using are indicated, as is the particular program he is running. The state of each user is indicated, whether he is running (RUN), not running (↑BS) or waiting for something. If a user has byped CTRL/S to stop his printing, ↑S will be printed; otherwise ↑Q will be printed. The amount of computer time used by each user since he logged in is given.

If more users are on-line than the system has core fields to hold them, the fact that the system is swapping is reported. The number of free core blocks used internally by the Monitor for Terminal buffering and various other purposes is printed. Then SYSTAT reports any unavailable devices, i.e., devices which are assigned to individual users. The job to which they are attached is also indicated. Finally, the number of available segments of disk storage is reported.

A sample SYSTAT listing is shown below. SYSTAT terminates by printing ↑BS and exiting to the Monitor.

```
.SYSTAT

STATUS OF TSS/8.24 DEC PDP-8 #1 AT 16:03:32 ON  9 DEC 74

UPTIME 00:03:32

JOB      WHO      WHERE      WHAT        STATUS      RUNTIME

 1       0, 3      K04       SYSTAT      RUN   ↑Q    00:00:02
 2      43,21      K00       PIP         KEY   ↑Q    00:00:10

AVAILABLE CORE  0K   FREE CORE=319

BUSY DEVICES    NONE

 284 FREE DISK SEGMENTS


↑BS
```

CHAPTER 9

PROGRAMS FOR HANDLING DATA

9.1 PERIPHERAL UTILITY TRANSFER ROUTINES (PUTR)

PUTR (Peripheral Utility Transfer Routines) is a utility program used to transfer files between all TSS/8 devices. PUTR can perform most of the functions of the programs PIP and COPY which operated under the previous version of the TSS/8 Monitor. The following is a list of TSS/8 devices used by PUTR.

Device Names

CDR:	Card Reader
Dn: or DTAn:	DECtape: Disk Monitor, COPY, OS/8, or PUTR
KBD:	Terminal Keyboard/Reader
LPT:	Line Printer
PTP:	High-Speed Paper Tape Punch
PTR:	High-Speed Paper Tape Reader
RKAn: or RKBn:	RKØ5: OS/8 file structure
SYS:	TSS/8 user file area
TTP:	Terminal Punch
TTY:	Terminal Printer

9.1.1 PUTR Commands

PUTR Commands are entered at the terminal in response to the asterisk which is printed at the left margin. A command may be typed with a command string of the form:

*command device:output files = device:input files

If a mistake is made while typing a PUTR command, there are three special characters which help correct the situation. Typing RUBOUT (or DELETE) deletes characters from the end of the line, printing the deleted characters. Typing a LINE FEED causes the command line to be printed on the user's terminal without executing it, and typing CTRL/U deletes the entire line.

Commands to PUTR can be abbreviated; however most cannot be abbreviated with less than three characters. PUTR commands and their purposes are listed below in alphabetical order.

COPY	Copy files.
DELETE	Delete files.
DIRECTORY	List directories of file structured devices.
EXIT	Returns to TSS/8 Monitor.
LIST	List files on the line printer (LPT).
PUNCH	Punches files on the high-speed punch.
TAPE	Punches files on the terminal punch.
TYPE	Types files on terminal (TTY).
ZERO	Zeroes RKØ5 to OS/8 format. Zeroes DECtape to PUTR format.

Examples of these commands are given in the following paragraphs.

Examples:

```
•R PUTR
*COPY DTAØ:FOO.SAV=FOO.SAV
```

This will copy the file FOO.SAV from the system area onto DTAØ.

NOTE

The DECtape mounted on DTAØ: must be either PUTR format or OS/8 format. PUTR cannot write onto the Disk Monitor or COPY format DECtapes.

To make a PUTR formatted tape use the ZERO Command which is described below. PUTR can only read Disk Monitor and COPY format DECtapes. This means that the user must convert any such tape that he may wish to write on. To convert a tape the user may use the following procedure. First, mount an empty tape on unit Ø. Then mount the tape to be converted on unit 1 and type the following:

```
.R PUTR
*ZERO DTAØ:
ARE YOU SURE?
YES
*COPY DTAØ:=DTA1:
```

This will move all the files from DTA1: (COPY or Disk Monitor format) to DTAØ: (PUTR format).

PUTR is a very powerful program, as these examples illustrate. An "*" in a command line says that any name will match.

```
*COPY DTAØ:=*.BAS
```

This command line will cause all files with the extension .BAS to be put onto the DECtape on unit Ø.

```
*DELETE *.TMP
```

The command line will delete all files on your system area with extension .TMP.

***DEL ???**

This command will delete all files whose names are three characters or less in length. The "?" is called a "wild card". This character will match on any character in its position.

***DEL ABD?**

This may be used to delete all files with names which are 3 or 4 characters in length and with ABD as the first three characters.

***DEL ***

This will delete all files on your system area. ZERO cannot be used for the system disk.

DEL DTA0:

This could be used if the DECtape on unit Ø were an OS/8 format tape and if the user wanted it to remain an OS/8 format tape. A ZERO command would change it to a PUTR format tape.

***TYPE ABC**

This will type the file ABC on the terminal. This file <u>is assumed</u> to be a TSS/8 format ASCII file.

***LIST DTA0:FILE.LS/OS8**

This will list an OS/8 format ASCII file on the line printer. OS/8 and TSS/8 character formats are different, and, therefore, character conversion switches have been added to PUTR.

NOTE

TSS/8 character files can be stored on OS/8 DECtapes and RKØ5 diskpacks. It is not necessary to do character conversions if the user is not going to use the file on OS/8. The following is an example of a command line which may be used if one wishes to use a file under OS/8.

```
*COPY RKAØ:*.PA/OS8=FILE/TS8
```

This will put FILE.PA on RKAØ:, converting it from TSS/8 format to OS/8 format. Switches on the output (left) side indicate how characters are to be packed. Switches on the input (right) side indicate how characters are to be unpacked.

NOTE

There is a compatability problem between TSS/8 and OS/8. OS/8 needs a CTRL/Z at the end of ASCII files. Since no TSS/8 programs insert this CTRL/Z, most OS/8 programs cannot work with ASCII files generated by TSS/8 and converted by PUTR.

```
*LIST FOO
```

is equivalent to

```
*COPY LPT:=FOO/TS8
*LIST FOO/BAS
```

This will list a BASIC program on the LPT.

```
*COPY FOO.BAS/BAS=PTR:
```

This will take an ASCII paper tape of a program and create a BASIC program file called FOO.BAS. When creating a BASIC file using

PUTR, the input must be valid and in increasing line-number order, or BASIC will not be able to process it.

NOTE

The Paper Tape Reader handler will print a "↑" and will wait for a carriage return to be typed to indicate the paper tape is ready. The default program names on PTR: and CDR: are "NONAME".

```
*COPY DTA0:FOO.PAL/TS8=CDR:/026
```

This will create a TSS/8 3-for-2 packed ASCII File from the card reader, DEC Ø26 card code.

NOTE

Ø26 is the default switch for the card reader.

```
*COPY PROG.BAS/BAS=CDR:
```

This will create a BASIC program file from the card reader (Ø26 code).

Additional notes:

Files on the TSS/8 system disk are unique only in their file names. Files on DECtape and RKØ5 are unique in file name and extension. This means that the user can have the files FOO.BAS and

FOO.BAC ON DECtape or RKØ5 but not on the TSS/8 system disk. Files may be copied from other user areas only if the user knows the name of the file. The following is an example of this.

```
*COPY DTAØ:MYFILE.BAS=HISFILE[1,12]
```

9.1.3. - Card Reader: END-OF-FILE, Hung Device

An end of file on the card reader is indicated by a card with a /* in columns 1 and 2, with the rest of the card blank.

If the card reader should have a hardware error, i.e., pick fail, or if it runs out of cards, the Monitor prints out the hung device message. When this happens, type "START". At this point, PUTR will ask "BAD CARD, TRY AGAIN?". The user may answer by typing either "N" followed by a carriage return to cause an end of file on the card reader, a carriage return only if it is desired to continue reading cards (after fixing the status of the card reader), or CTRL/C to give up. The error message above will also print out if 39,40, or 80 columns were not read. If this happens, place the card which was just read at the bottom of the deck to be re-read.

9.2 - COPY COMMAND

The general format for the COPY command is:

*COPY [dev:] output name [.extension] [/switches] [=] [dev:]
input name [.ext] [P, Pn] [/switches]

Items in large brackets are optional. One of "← ", "=", or "<" must be used to separate the output and input file descriptors. Up to six input file descriptors may be specified (separated by commas). The default output and input devices (dev:) are SYS:. The default output and input extensions are * (to be explained).

The switches on the output side indicate the type of character packing (default is IMAGE). Switches on the input side indicate the type of character unpacking (default is image). "[P,Pn]" is the account number where the file is located. This is only applicable to input from SYS:. If a device is specified without a file name, all the files on that device will be selected. The asterisk (*) can be used either for the file name or the extension, or both. If used on input side, all file names or extensions will match.

NOTE

A user may not reference another user's file directory. Therefore, asterisks (*) and wild cards (?) cannot be successfully used when an account number is given.

The asterisk on the output side uses the file name or the extension of the file opened on the input side.

CAUTION

TSS/8 system files are unique in name only, whereas DECtape and RKØ5 files are unique in name and extension.

It is possible to have the files FOO.BAS and FOO.BAC on a DECtape or RKØ5 but not under the same account number on the system disk. Storing FOO.BAS on the system disk will delete any file named FOO, for example FOO.BAC.

The question mark (?) is a wild card character which is similar to the asterisk (*), except that it matches any character in its position. The question mark cannot be used on TSS/8 extensions since these extensions are internal numbers rather than character extensions.

Examples of COPY Commands:

***COPY FOO=DØ:FOO.BAS**	Copies FOO.BAS from DØ: to SYS: (same extension).
***COPY *=DØ:*.BAS**	Copies all files with .BAS extensions from DØ: to SYS:
***COPY *=DØ:A,B,C**	Copies A,B,C from DØ: to SYS: (same name and extension).
***COPY *.TMP=DØ:*.ASC**	Copies all files from DØ: with the extension .ASC to files on SYS: with same name but with the extension .TMP.
***COPY *.ASC/TS8=DØ:FOO.AS/OS8**	Converts an OS/8 character format to TSS/8 format while copying a file from DØ: to the system disk, SYS:.
COPY DØ:=	Saves all files from SYS: onto DØ:

NOTE

If DECtape is used for output, it must be either a PUTR structure or OS/8 type tape, and not a Disk Monitor or COPY tape.

9.1.1.1 ZERO Commands

Format

*ZERO Dn: or *ZERO DTAn:	Zeroes DECtape directories to PUTR file structure.
*ZERO RKAn: or *ZERO RKBn:	Zeroes RKØ5 to OS/8 file structure.

After typing a ZERO command, PUTR will ask "ARE YOU SURE?" Type Y or N for YES or NO, as the case may be.

9.1.1.2 DELETE Command

Format

*DELETE dev: name ext. , more files

NOTE

If the user wishes to zero his account he may type the following command line.

***DELETE ***

The following command line may be used to zero OS/8 DECtapes and to preserve the OS/8 file structure.

*DELETE Dn: or

*DELETE DTAn:

9.1.1.3 DIRECTORY Commands

Format

*DIRECTORY [dev:] name [.ext] or
*DIRECTORY [dev:]

Example:

*DIRECTORY (carriage return)

This command line will give a directory listing of all files on your disk area (SYS:).

***DIR DØ:/LPT**

This command will give a directory listing of the DECtape on unit Ø. The listing will be printed on the line printer, if it is available.

Please note that directory listings of the Monitor, and COPY format tapes will include just the names and extensions.

```
*DIR *.BAS
```

This will make a directory listing of the files on SYS: that have the extension .BAS.

9.1.1.4 LIST Command

Examples:

```
*LIST FOO
*LIST BASPRO/BAS
*LIST D0:LIST.LS/OS8
```

The LIST command changes the default input switch from /IM to /TS8 and sets in the default output device, LPT:. "LIST FOO" is identical to "*COPY LPT:=FOO/TS8".

9.1.1.5 TYPE Command

The TYPE command is identical to the LIST command except that the output device is TTY:. "LIST" and "TYPE" will insert a carriage return/line feed in the output when the line length for the LPT: or TTY: is exceeded and will not output RUBOUTs. "TYPE" will insert 4 line feeds for every form feed it sees. If the text that exceeds the line limit for the device need not be printed, then the /TRIM switch should be appended to the file descriptor. For example,

```
*TYPE FOO/TRIM
```

9.1.1.6 PUNCH Command

The PUNCH command "*PUNCH name" is equivalent to:

*COPY PTP:=name/TS8

9.1.1.7 TAPE Command

The TAPE command "*TAPE name" is equivalent to:

*COPY TTP:=name/TS8

Both PUNCH and TAPE will punch LEADER/TRAILER on the tapes. The following are examples:

***PUNCH FOO**	(TSS/8 ASCII Format)
***PUNCH BASPRG/BAS**	(BASIC Format)
***PUNCH SAVPRG/SAV**	(SAVE Format)

9.1.1.8 EXIT Command

This is the command which will enable the user to exit from PUTR.

Example:

```
*EXIT
↑BS
```

or

```
*E
↑BS
```

9.1.1.9 Special Notes

BINARY PAPER TAPES

No switch is required to punch a binary tape. However, if the binary is an OS/8 file /OS8 will be required. Binaries on TSS/8 and

OS/8 are stored the same way that ASCII files are stored on their respective systems.

When a binary tape is read use the /TS8 switch or the /OS8 switch on the output side of the COPY Command, and /BIN on the input side. /BIN tells the input to compute a checksum on the tape.

Examples:

```
*COPY TSFILE.BIN/TS8=PTR:/BIN
*COPY D0:OSFILE.BN/OS8=PTR:/BIN
```

SAVE FORMAT PAPER TAPES

Punch the tape, with /SAV on the input file descriptor, using PUNCH or TAPE.

Example:

```
*PUNCH SAVFIL/SAV
```

When reading the tape, read it with /SAV on the input side.

Example:

```
*COPY FOO.SAV=PTR:/SAV
```

/SAV and /BIN should not be mixed on the same command line.

Example of bad switch use:

```
*PUNCH FOO/SAV,FOO2/BIN
```

NOTE

Typing CTRL/C will cause PUTR to return to the "*". If PUTR is in the process of copying files, the file that is currently being transferred will not be completely transferred. Therefore, the user should delete the last name which PUTR printed.

9.1.1.10 Default Propagation

When more than one file description is included on the input side of a PUTR command, then any parameters not specified in one file description are the same as those of the previous description. Example:

```
*LIST DTA0:A,B,C
```

This will list the files A, B, and C from DTAØ.

```
*LIST A,DTA0:B,DTA1:C
```

This will list the files A from SYS: (the user's file area), B from DTAØ:, and C from DTA1. A similar rule also applies to switches. To list two OS/8 files from DTAØ:, do NOT type

```
*LIST DTA0:A,B/OS8
```

for the /OS8 will apply only to file B. Rather, type

```
*LIST DTA0:A/OS8,B
```

The /OS8 will now apply to both file A and file B.

Using CTRL/O

While PUTR is transferring files, it prints the name of each input file as it begins reading it. This print-out can be disabled by typing CTRL/O. The print-out can be re-enabled by typing a carriage return.

9.1.1.11 DECtapes and RKØ5 Disk

DECtapes can be Disk Monitor, COPY, OS/8, or PUTR file structure tapes. PUTR will know which tape is mounted. PUTR will not write on the Disk Monitor or COPY DECtapes. However, it will read them so that the user may update his tapes if necessary. The RKØ5 is in OS/8 file structure format. RKØ5 is divided into two logical devices, RKAn: and RKBn: where n is the drive number (Ø-3). RKAn: is the first half of the disk, and RKBn: is the second half of the disk.

Each of the writeable devices Dn:, RKAn:, and RKBn: has directory space for up to 240 entries. A ZERO operation on DECtapes zeroes to PUTR format. A ZERO operation on RKAn: or RKBn: zeroes to OS/8 format.

NOTE

The data format of the files is independent of the file structure. This means that TSS/8 files can be put on an OS/8 type device without any conversion. Any ASCII or binary file that will be used by the OS/8 or TSS/8 systems should have the appropriate switches (/OS8 or /TS8) while copying. It is also helpful when storing both formats (/OS8 and /TS8) on the same device, that you use TSS/8 type extensions (.BIN, .ASC, .BAS) for the TSS/8 type files and OS/8 type extensions (.BN, .AS, .BA...) for the OS/8 type files.

9.1.2 TSS/8 FILE EXTENSIONS

File Extension	Internal Number	
.ASC	1	ASCII
.SAV	2	SAVE
.BIN	3	BINARY
.BAS	4	BASIC SOURCE
.BAC	5	BASIC COMPILED
.FCL	6	FOCAL SOURCE
.TMP	7	TEMPORARY
.	10	Blank - required by System.
.DAT	11	DATA FILE (BASIC)
.LST	12	LISTING FILE
.PAL	13	PAL SOURCE
.	14	UNASSIGNED
.	15	UNASSIGNED
.	16	UNASSIGNED
.	17	UNASSIGNED

9.1.3 PUTR SWITCHES

Switches for all devices:

IM - IMAGE transfer; no conversion
6BIT - Six Bit ASCII packed 2 characters/word
X240 - Excess 240, 2 characters/word
X237 - Excess 237, 2 characters/word
OS8 - OS/8, 3 characters/2 words
TS8 - TSS/8, 3 characters/2 words
BAS - TSS/8 BASIC format

Paper Tape Switches

BIN - Binary check summing
SAV - Save format with check summing

Card Reader Switches

ALP - RCRA, 6 bit internal alpha.
COM - RCRC, compressed 8 bit (PDP8-E systems)
Ø26 - Ø26 to ASCII
Ø29 - Ø29 to ASCII

Line Printer and Terminal

TRIM - Trims off excess line.
(NOTE: a carriage return /line feed will be inserted in a line that is too long for the device if TRIM is not specified.)
LPT - List on line printer if available (DIRECTORY command only).

9.2 PERIPHERAL INTERCHANGE PROGRAM (PIP)

Disk is a convenient storage medium for many files; however, it may be more useful to keep some programs on paper tape. PIP provides a convenient means of transferring files between disk and paper tape, for those users who wish to preserve copies of their files off-line.

9.2.1 PIP Conventions

PIP may be considered a link between disk file storage and paper-tape devices. To punch a desired file, PIP obtains that file from the disk and punches it on paper tape. Similarly, to load a paper tape, PIP inputs the tape from the reader, then outputs it to a disk file.

The way files are named is important to PIP. Files on disk are always named. Paper tapes, on the other hand, have no names as far as the system is concerned (although the user can label the physical tape in any manner he chooses). Paper tapes never have file names; therefore, PIP uses the absence of a file name to indicate a paper tape (absence of a file name is indicated by striking the RETURN key).

The way in which INPUT: and OUTPUT: is indicated provides the means for determining the direction of file transfer. If PIP is to get its input from the disk, the input is a file name; if the input is from a paper tape, no file name is given. Similarly, if PIP is to output to the disk, the file name is indicated; if output is to paper tape, no name is given. To call PIP, type:

```
.R PIP
```

9.2.2 Paper Tape to Disk Transfers

To move a paper tape to disk, strike the RETURN key when PIP requests INPUT: Since PIP must output to the disk, respond to OUTPUT: by typing a file name. When PIP requests OPTION: type T to indicate that the paper tape is being loaded from the Terminal reader. For example:

```
.R PIP

INPUT:
OUTPUT:FILE1
OPTION:T
```

The paper tape in the low-speed reader is read and stored in the system as FILE1.

9.2.3 Disk to Paper Tape Transfers

To move a disk file onto paper tape, the use of file names is reversed since PIP must input a disk file and output it to paper tape. The option remains the same. For example:

```
.R PIP

 INPUT:FILE1
OUTPUT:
OPTION:T
```

The contents of FILE1 are then punched at the Terminal.

9.2.4 High-Speed Reader/Punch Assignments

PIP can also be used with high-speed paper-tape devices. The format of the INPUT: and OUTPUT: responses is the same. However, for the high-speed reader, the option is R and for the punch it is P.

Since the reader and punch are assignable devices, they are not always available (other users may have one or both assigned). Therefore, whenever PIP is given a command which utilizes one of these devices, it checks to make sure that the device is available. If it is, PIP automatically assigns it (thus, it is not necessary to assign the device before running PIP). If the device is unavailable, PIP informs the user. For example:

```
  INPUT:
OUTPUT:ABCD
OPTION:R
```

PIP reads the paper tape in the high-speed reader and stores it in the system as ABCD.

```
   INPUT:ABCD
  OUTPUT:
  OPTION:P
```

PIP punches out file ABCD on the high-speed punch.

```
    INPUT:ABCD
   OUTPUT:
   OPTION:P
   DEVICE NOT AVAILABLE OR HUNG
```

The punch is assigned to another user, or there is no punch on the EduSystem 50, or there is one but it is turned off.

9.2.5 Bin Format File Transfers

The examples above work for all ASCII file transfers (except BASIC programs, explained in Section 9.4.8). The examples are also valid for punching BIN files, with one exception; most installations do not allow any BIN format tapes to be loaded from the low-speed reader.

9.2.6 Moving Disk Files

PIP can be used to move the contents of one file into another. This is often useful in copying a file from another user's library (providing the file is not protected) into your own library. To copy from disk file to disk file, specify a file name for both input and output. Reply to OPTION: by striking the RETURN key. For example:

```
 INPUT:FOCAL 2
OUTPUT:FOCALX
OPTION:
```

PIP gets FOCAL from account number 2's library and moves it into the file FOCALX.

9.2.7 Deleting Disk Files

One of the principal reasons for punching files on paper tape is to free disk space. Once punched, the disk file is no longer needed. PIP offers a convenient means of deleting files, the Delete option:

```
 INPUT:ABCD
OUTPUT:
OPTION:D
```

PIP deletes file ABCD, provided that the file is not protected against being changed.

9.3.7 BASIC File Transfers

BASIC stores its programs in a unique file format. Therefore, it is not possible to load or punch BASIC files in the usual way. To provide a convenient means of handling BASIC programs, the B option is available in PIP. The B option is used for both reading and punching BASIC programs. The responses to INPUT: and OUTPUT: indicate the direction of the transfer; the high-speed reader or punch is always assumed for the B option. (To read or punch tapes at low-speed, use BASIC itself.)

PIP assumes that any BASIC tapes it loads are clean and error-free. Only tapes actually created by BASIC should be loaded with PIP. Tapes created off-line, and thus liable to contain errors, should be loaded low-speed by BASIC itself with the TAPE command.

9.3.8 Save Format File Transfers

Another special file format is that of the SAVE files, those programs directly executed by EduSystem 50. (The Systems Library Programs are examples of SAVE format files.) PIP provides the S option, to allow these files to be punched on paper tape. SAVE format tapes make sense only to PIP and PUTR. They cannot be input to any other Systems Programs.

The responses to INPUT: and OUTPUT: indicate the direction of the transfer; the high-speed reader or punch is always assumed for the S option.

NOTE

SAVE format tapes include a checksum. If PIP detects an incorrect read, it prints LOAD ERROR, and terminates the load, repeating the request for input.

TABLE 9-1

PIP OPTION SUMMARY

Option	Explanation
B	Transfer a BASIC program file between the disk and the high-speed reader or punch. The response to input and output indicates the direction of the transfer.
D	Delete the file specified for input.
F	List a BASIC program on the line printer.
K	Load a save format paper-tape from the terminal. This option will not work on most TSS/8 installations.
L	Transfer an ASCII file from the disk to the line printer.
P	Punch the contents of a disk file on the high-speed punch.
R	Read a tape from the high-speed reader and store it as a disk file.
S	Transfer a SAVE format file between the disk and the high-speed reader or punch. The response to INPUT: and OUTPUT: indicates the direction of the transfer.
T	Transfer a file between the disk and the terminal reader or punch. The response to INPUT: and OUTPUT: indicates the direction of the transfer.

9.3 COPY PROGRAM

Many EduSystem 50 installations include one or more DECtapes. For these installations, DECtape provides a convenient and inexpensive means of file storage. The COPY program is used to transfer files between disk and DECtape.

9.3.1 Using and Calling COPY

COPY is the intermediary between disk and DECtape. To write a disk file out to DECtape, COPY inputs the file from the disk, then

outputs it to the DECtape. To bring a DECtape file onto the disk, COPY inputs from the DECtape, then outputs to the disk.

Files kept on DECtape have file names just as they do on the disk. To avoid confusion, the user must tell COPY where the file is to be found. If it is on DECtape, the DECtape designation and the number of the DECtape unit must preface the file name. The DECtape number is always separated from the file name by a colon. Thus D1:FILE1 means the file name FILE1 on the DECtape which is currently mounted on DECtape unit number one. The number of available tape units varies among installations. The maximum is eight (numbered 0-7). If a file name is not prefaced by a DECtape number, the file is assumed to be on the system disk.

Files stored on DECtape do not have protection codes in the sense that disk files do. They are, however, protected against unauthorized access. When a DECtape is not mounted, it is not available to any user. When it is mounted, it is available only to the user who has assigned the DECtape unit on which it is mounted. Even then it can not be altered unless the DECtape unit is set to WRITE ENABLE. Users should be sure to assign a DECtape unit before mounting their tape, and dismount the tape before releasing the device. Normally, the DECtape unit to be used should be assigned before calling COPY.

To call COPY, type:

```
.R COPY
```

COPY responds by asking which option the user wishes to employ. The COPY options are discussed below and summarized in Table 9-2.

9.3.2 Loading Files from DECtape

To load a file onto the disk from DECtape, use the COPY option. When COPY requests OPTION, respond with COPY, or C, or strike the RETURN key (the COPY option is assumed). When COPY requests INPUT, type the number of the DECtape unit on which the file can be found (D0, D1, D2, D3, D4, D5, D6, or D7) followed by a colon and the name of the file on the DECtape. When COPY requests OUTPUT, type and enter the name to be given to the output file on the disk. COPY then moves the DECtape file onto the disk. (When using COPY, it is not mandatory to insert a space between the device designator and the device number.) For example:

```
OPTION- COPY
INPUT- D4:PQR
OUTPUT- PQR
```

If for any reason, COPY cannot find the DECtape file specified for input (the specified DECtape is unavailable or nonexistent, or the file name does not exist on that DECtape), COPY prints a ? and repeats the request for input. If the disk file specified for output already exists, COPY prints a ? and repeats the request for output. COPY does not overwrite an existing file. For example:

```
OPTION- C
INPUT- D9:PQR
?INPUT- D4:PQRS
?INPUT- D4:PQR
OUTPUT- PQR
```

9.3.3 Saving Disk Files on DECtape

Saving a disk file on DECtape is very similar to loading one. The option is still COPY. For input, respond with the name of the file on the disk. For output, type the DECtape unit number, colon, and the name to be given to this file. For example:

```
OPTION- C
INPUT- ABCD
OUTPUT- D4:ABCD
```

If COPY cannot find the file on the disk, or if it is protected, COPY prints a ? and repeats the request for input. If COPY cannot create the desired DECtape file (the specified DECtape does not exist or is unavailable, or it is not WRITE ENABLED, or a file by that name already exists on the tape) COPY prints a ? and repeats the request for output.

9.3.4 Listing Directories

COPY can be used to list the directory of a device. To list a directory, respond to OPTION by typing LIST, or just L. COPY then asks which device directory it is to list. To list a DECtape directory, respond with the device name (D0,...,D7). Do not follow it by a colon. For example:

```
.R COPY

OPTION- LIST
INPUT- D1

1372. FREE BLOCKS

NAME       SIZE     DATE
BASIC .SAV   66    9-MAR-70
FACTAL.BAS   10    2-MAR-70
CONVER.BAC    6    1-MAR-70
PALD  .SAV   32   31-MAR-70
```

The unit of DECtape storage is the block, which is 128 (decimal) words. Because the unit of disk storage, the segment, is generally 256 words, a file occupies twice as many blocks of DECtape storage as it did segments on the disk.

COPY can also be used to list the user's disk directory. Use the LIST option, but respond to DEVICE by simply striking the RETURN key. The directory listing is similar to the listing obtained by running the CAT program.

9.3.5 Deleting Files

COPY can also be used to delete files, either on the disk or on a selected DECtape. To delete a file, respond to OPTION by typing DELETE, or just D. Respond to INPUT by typing the name of the file to be deleted. If the file is on a DECtape, preface the file name with the DECtape unit number and a colon. For example:

```
OPTION- DELETE
INPUT- D4:ABCD
```

If COPY cannot find the file to be deleted, or having found it, cannot delete it (it is a protected disk file or a DECtape file on a unit which is not WRITE ENABLED), COPY prints a ? and repeats the request for INPUT.

9.3.5.1 Deleting All Existing Files on a Device

COPY can be used to delete all existing files on a device. To do so, respond to OPTION by typing ZERO, or just Z. When COPY requests INPUT respond with the name of the device. COPY will not zero the system disk. The ZERO option should also be used to format a blank DECtape before attempting to copy any files onto it. For example:

```
OPTION- ZERO
INPUT- D4
```

TABLE 9-2

COPY OPTION SUMMARY

Option	Abbreviation	Explanation
COPY	C	Transfer a file between disk and DECtape.
DELETE	D	Delete a file.
EXIT	E	Exit to the Monitor.
LIST	L	List a directory.
ZERO	Z	Delete all files.

COPY cannot delete files from a DECtape unless it is WRITE ENABLED. It cannot delete disk files which are write protected.

9.3.6 Example of COPY Usage

```
•ASSIGN D 5
D 5 ASSIGNED
•R COPY

OPTION- ZERO
DEVICE- D5

OPTION- LIST
DEVICE- D5

1462. FREE BLOCKS

NAME      SIZE    DATE

OPTION- LIST
DEVICE-

DISK FILES FOR USER 54,40 ON  9-DEC-74.

NAME      SIZE  PROT    DATE
SOLVE .BAS   1    12    9-DEC-74

TOTAL DISK SEGMENTS: 1
```

```
OPTION- COPY
INPUT- SOLVE
OUTPUT- D5:SOLVE


OPTION- DELETE
INPUT- SOLVE


OPTION- LIST
DEVICE- D5



1460. FREE BLOCKS

NAME       SIZE     DATE
SOLVE .BAS    2    9-DEC-74



OPTION- LIST
DEVICE-


DISK FILES FOR USER 54,40 ON  9-DEC-74.

NAME       SIZE  PROT    DATE

TOTAL DISK SEGMENTS: 0

OPTION- COPY
INPUT- D5:SOLVE
OUTPUT- ABCD


OPTION- LIST
DEVICE-


DISK FILES FOR USER 54,40 ON  9-DEC-74.

NAME       SIZE  PROT    DATE
ABCD  .BAS    1    12    9-DEC-74

TOTAL DISK SEGMENTS: 1

OPTION- RENAME
INPUT- ABCD
OUTPUT- FIE.BIN<17>


OPTION- LIST
DEVICE-


DISK FILES FOR USER 54,40 ON  9-DEC-74.

NAME       SIZE  PROT    DATE
FIE   .BIN    1    17    9-DEC-74

TOTAL DISK SEGMENTS: 1
```



```
OPTION- EXIT
↑BS
.RELEASE D 5
```

CHAPTER 10

ADVANCED MONITOR COMMANDS

10.1 INTRODUCTION

The fundamental Monitor commands described previously are those needed to utilize existing System Library Programs. The EduSystem 50 Monitor also provides powerful commands for users who wish to create their own library programs.

To use the System Library Programs described previously, it was not necessary to be familiar with the actual machine that runs them, the PDP-8/E. To create new library programs for EduSystem 50, this is necessary because they are written in the PDP-8 assembly language. The user codes his programs for a 4K PDP-8, subject to the time-sharing conventions discussed in this section. The programs are created with EDIT, then assembled by PAL-D and loaded by LOADER. Only at this point are the programs able to be run by EduSystem 50. In the course of this program development, the same program exists in many formats.

The source program is a disk file containing ASCII characters in an Editor format. PAL-D reads the file and translates it into a second file, the assembled program in BIN format. Neither of these files is capable of being executed directly by EduSystem 50. The BIN format tape must be loaded into core by LOADER before it can actually be executed.

At this point it is possible to save the program in a file format that is directly executable by EduSystem 50. Such a file, called a SAVE format file, contains an image of the user's core area after the program has been loaded by LOADER. These SAVE format files differ

from all the files which are created by System Library Programs and cannot be executed directly by EduSystem 50. Thus, it is not possible to save a BASIC program (e.g., FILE1 while running BASIC), then return to Monitor type R FILE1, and get meaningful results. The program in FILE1 must be executed under control of the BASIC language processor. Only SAVE format files can be called into execution directly by the R command. (All System Library Programs are stored in SAVE format and can be run with the R command.)

NOTE

In the following examples, Sn, Cn, and Dn are used to stand for alphanumeric strings (such as file names), octal numbers, and decimal numbers, respectively.

A number of Monitor command conventions are available to make the commands easier to use. First, more than one command may be typed on a line. Individual commands are separated by a semi-colon (;). Second, only enough characters of a command to uniquely specify it need by typed. Thus, DEPOSIT can be abbreviated DE or DEP.

```
•LOAD FILE1;DEP 20 7000;ST 200
```

is exactly equivalent to:

```
•LOAD FILE1
•DEPOSIT 20 7000
•START 200
```

These conventions are available for the elementary Monitor commands as well. They are, however, especially convenient for the advanced commands.

10.2 CONTROL OF USER PROGRAMS

Once a PAL-D program has been loaded by LOADER, several Monitor commands are available for controlling its execution. These commands are shown in Table 10-1.

It is possible to give these utility commands while a user program is running. The CTRL/B character (↑B) gets the attention of the Monitor without stopping program execution. (↑B followed by the S command stops the program.) ↑B can be used together with the WHERE command to follow program execution. After executing these commands, Monitor does not put the Terminal back into Monitor mode.

TABLE 10-1

MONITOR PROGRAM CONTROL COMMANDS

Command	Explanation
DEPOSIT C1 C2...Cn	Deposit the octal values C2 to Cn in the locations starting at C1. DEPOSIT is used to make small octal modifications to a user program. No more than 10(decimal) locations can be modified by a single DEPOSIT instruction.
EXAMINE C1	Print the octal contents of location C1.
EXAMINE C1 D1	Print the contents of D1 locations starting at C1.
START	Restart execution of a user program where it was interrupted (either by execution of an HLT or by ↑BS typed at the keyboard). When the START command is given, the program's state is restored.
START C1	Start execution of a user program at location C1. When a program is started, keyboard input is no longer interpreted as commands to Monitor. Input characters are passed to the running program. START C1 clears the user's AC and link.

Table 10-1. (Cont.)

Command	Explanation
WHERE	Print the present status of the user program. The user's AC, PC, LINK and switch register are printed. Also, any EAE registers which are present are printed.

10.3 DEFINING DISK FILES

The Monitor allows the user to save core images of his program on the disk for future use. However, before saving such a core image, the user must define a disk file in which to save it.

Disk files, like the user's core, are made up of 12-bit words. Unlike the user's core, which is always 4K in size, a file can be any size. The unit of disk file storage is the segment; a segment is 256 (decimal) words long. Files are at least one segment long when created and grow by appending additional segments to the end of the file. In defining a file, the user first creates it, then extends it to whatever length he needs. To have a whole 4K image on a system with a segment size of 256 (decimal) words, a 16 segment file is required. If only part of the contents of the user's core is to be saved, a correspondingly smaller file can be used.

A file can be created at any time. However, to modify or redefine it in any way, the file must be open. Up to four files can be open for a user simultaneously. Opening a file connnects it to an internal open file number (0, 1, 2, or 3). Once a file is open, it is referenced by this internal file number rather than by its file name.

10.3.1 Creating a Disk File

The CREATE command defines an area of disk space and associates it with the name given in the command line.

The file name can be one to six alphanumeric characters of which the first must be a letter. Creating a file deletes any existing file of the same name, unless that file is write protected. When created, files are always one segment in size. A new file is arbitrarily assigned a protection code of 12, meaning that other users may access it but only the owner may change it. Until it has been written, the contents of a newly defined file are undefined.

10.3.2 Opening and Closing a File

To use a file, it must first be opened with the OPEN command. A file can be opened on any of four internal file numbers: 0, 1, 2, or 3. A user can have up to four files open at a time. If a file is open on an internal file number for which a file is already open, that file is first closed. For example:

```
•CREATE AB
•OPEN 1 AB
```

AB is now an open file and can be referenced as file 1.

An open file can be closed at any time by means of the CLOSE command. Once closed, a file cannot be accessed in any way until it is reopened. It is possible to close more than one file with a single command. For example:

```
•CLOSE Ø 1 2 3
```

10.3.3 Extending, Reducing, and Renaming a Disk File

When created, a file is one segment long. If a larger file is needed, the original file can be extended. For example, the command:

.EXTEND C1 D1

extends the file presently open on internal file C1 by D1 segments. Extending a file adds one or more segments to the end of that file. The contents of the old part of the file are not changed. Until written, the contents of the newly added segments are unspecified. An existing file may be reduced in size by means of the REDUCE command. For example, the command:

.REDUCE C1 D1

reduces the file presently open on internal file C1 by D1 segments. Reducing a file deletes the number of segments indicated from the end of the file. The contents of remaining segments of the file are unchanged. If a file is reduced to zero segments, or if D1 is greater than the number of segments in the file, it is deleted entirely. An example of the creation and deletion of a 4K file:

```
•CREATE FOURK
•OPEN 3 FOURK
•EXTEND 3 15
•REDUCE 3 16
```

Existing opened files can be renamed. Renaming a file does not change its contents in any way. For example, the command:

•RENAME C1 S1

renames as S1 the file open on internal file number C1.

10.3.4 Protection Codes

The user can protect his files against unauthorized access. He can also specify the extent of access certain other users can have to his files. For example, a user's associates can be permitted to look at the data of certain files but not permitted to alter that data.

When it is created, a file is assigned a protection code of 12. This protection code is defined below and can be changed , but only by the owner of that file. For example, the command:

.PROTECT C1 C2

gives the protection code C2 to the file open on internal file number C1.

The protection code is actually a 5-bit mask. Each bit specifies a unique level of protection.

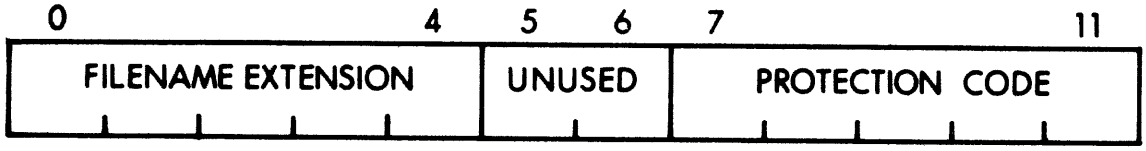
File protection masks (C2) are assigned as follows:

1	Read protect against users whose project number differs from owner's.
2	Write protect against users whose project number differs from owner's.
4	Read protect against users whose project number is same as owner's.
10	Write protect against users whose project number is same as owner's.
20	Write protect against owner. To change the program the owner must change the protect code.

Protection codes are determined as the unique sum of any of the above codes. Some of the more common protection codes are as follows:

Command	Explanation
PROTECT 1 0	Allow other users to access the file and change it.
PROTECT 1 12	Allow other users to access the file but not change it.
PROTECT 1 17	Allow only the file owner to read the file. He can also change it.
PROTECT 1 37	Allow only the file owner to read the file. He cannot, however, change it. (To change it, he must first change the protection.)

A user's project number is the first 2 digits of the 4-digit account number. The protection word contains a filename extension, and has the following format:



The filename extension gives additional information about the file, which is printed in some directory listings. The filename extension codes are:

0	blank	
1	.ASC	ASCII files, such as PAL source.
2	.SAV	Save format files.
3	.BIN	Binary files; must be loaded with program LOADER.
4	.BAS	BASIC source file.
5	.BAC	BASIC compiled program file.
6	.FCL	FOCAL file

7	.TMP	Temporary file.
11	.DAT	Basic data file.
12	.LST	Listing file.
13	.PAL	PAL source file.

The protection word may be set by using PROTECT:

```
.PROTECT 0 0217
```

This changes the protection of the file open under internal file number 0 so that the file has an extension of .ASC, and that it cannot be read or written by any person other than its owner.

Finally, the user can ask what file is open on a given internal file number by means of the F (File information) command. For example, the command:

.F C1

prints the following information about the file presently open on an internal file C1:

a. Account number of file owner.
b. Name of file.
c. File extension.
d. Protection code.
e. Size of file in segments (decimal).

For example:

```
.F 1
0002 TYPE 01 12 5
```

10.3.5 Error Conditions

There are a number of error conditions which prevent the execution of the file definition commands (as previously described). One of the following error messages is printed by Monitor if an error condition is detected:

Message	Explanation
DIRECTORY FULL	A CREATE command has been issued, but the user's directory is full. He can delete any of his files to make room for the new file. This message also indicates a user has exceeded his disk quota.
[MYFILE EXCEEDING DISK QUOTA]	The user has extended a file beyond his allowed disk quota. The amount of extra space (grace) the user is allowed is determined by the system manager.
FAILED BY n SEGMENTS	The user has attempted to extend a file, but cannot because of lack of segments on the system, or because he is attempting to go beyond the grace quota. The number of segments requested, but not available, is printed.
FILE IN USE	An EXTEND, REDUCE, PROTECT, or RENAME command has been issued for a file which is in use elsewhere by another user. Because changing a file which is being used (i.e., has been opened) could disrupt another user's work, under these conditions such a change is prohibited.
FILE NOT FOUND	The user has attempted to OPEN a non-existent file.
FILE NOT OPEN	An EXTEND, REDUCE, PROTECT, or RENAME command has been issued for an internal file number for which no file is open.
PROTECTION VIOLATION	An attempt has been made to change a file which is write protected against the user.

10.4. SAVING AND RESTORING USER PROGRAMS

Once a file has been defined, the user can save all or any part of his user core in the file. Files and user core are addresses in

the same way, by 12-bit words. The user can transfer his file into any part of core.

The SAVE command requires one to five parameters. The name of the file to be written into must always be given. If the file is not in the user's own library, the appropriate account number is entered before the file name. (Writing into a file owned by another user is subject to file protection.) In either case, the parameters are separated by spaces. The SAVE command writes the indicated section of core out into the indicated file.

If no parameters follow the file name, Monitor starts at location zero of the user's core and saves it in location zero of the disk file. It continues to write core locations into the disk file until: (a) it has written the whole 4K or (b) it has filled the file. Either condition completes the SAVE.

The user can further define his SAVE command by indicating parts of core to be saved in specific parts of the disk file. He does this by typing one to three parameters following the file name. The first parameter following the file name indicates a specific disk file address at which to begin writing. The second parameter following the file name indicates a specific core address at which to begin the transfer. If only the first two parameters are typed, the transfer terminates when either the end of core or the end of file is reached.

Command	Explanation
SAVE S1 SAVE C1 S1	Assuming that a disk file S1 exists, and that it is not write protected, the contents of core are saved in S1. In the first case, S1 is assumed to be in the library of the user giving the command. In the second case, it is assumed to be in the library of the user whose account number is C1.
SAVE S1 C2 C3 C4	Locations C3 to C4 (inclusive) are saved in file S1 starting at disk file location C2. S1 is assumed to be in the user's own library. If S1 is preceded by the parameter C1, it is assumed to be in the library of the user whose account number is C1.

Once a core image has been saved in a disk file, it can be restored to core by means of the LOAD command. It should be noted that the Monitor command LOAD is very different from the System Library Program LOADER. LOADER loads a BIN format file (created by PAL-D) into the user's core. LOAD loads a SAVE format file (created by a previous SAVE command) into core.

The LOAD command requires from one to five parameters. The name of the file to be loaded must always be given. If the file is in the user's own library, this file name is typed after the SAVE command itself. If it is in another user's library, his account number is entered before the file name. (Reading another user's file is subject to file protection.) In either case, the parameters are separated by spaces.

Command	Explanation
LOAD S1 LOAD C1 S1	Assuming that a disk file S1 exists, and that it is not read protected, the contents of the file S1 are loaded into core. In the first case S1 is assumed to be in the library of the user giving the command. In the second case, it is assumed to be in the library of the user whose account number is D1.

The user can further define his LOAD command by using the same optional parameters discussed in the section on the SAVE command.

Command	Explanation
LOAD S1 C2 C3 C4	Locations C3 to C4 (inclusive) are loaded from the file S1 starting at file location C2.
LOAD NEWF 5 10 17	Words 5 to 14 (inclusive) of the file named NEWF are loaded into locations 10 to 17 of the user's core.

It is not necessary to open a file before using it in a LOAD or SAVE command. Both commands automatically open the specified file on internal file number 3 before performing the transfer. After completion of the command, the file remains open on file number 3.

A special macro-command, RUN, exists to allow a program to be loaded and started all in one command.

Command	Explanation
RUN S1 RUN C1 S1	Load file S1 into core from the disk and start execution at location 0. In the first example, file S1 is assumed to be in the user's own library. In the second, it is assumed to be in the library of the user whose account number is C1. RUN S1 is exactly equivalent to LOAD S1; START 0. RUN C1 S1 is exactly equivalent to LOAD C1 S1; START 0.
RUN S1 C2 RUN C1 S1 C2	Load file into core from the disk and start execution at location C2.

The R command (see the section EduSystem 50 Monitor) is a special case of the RUN command. For example, the command:

.R S1

loads file S1 from the System Library (account number 2) and starts at location 0. R S1 is exactly equivalent to RUN 2 S1.

Typing an address after the program name in a "R" or "RUN" command causes execution to begin at that address, rather than at zero.

Sometimes, when typing a complex SI command, the rubout (or delete) key may have been used a number of times to make corrections, and the user may not be quite sure of what was typed. A LINE FEED may be typed to instruct SI to print out the command line as SI currently understands it. If the command line was entered while a program was running by prefacing the command with ↑B, SI will print ↑B to signify this, or else it will start the line with a dot. Similarly, if an attempt is made to rubout characters when there are none to rubout, SI will print either ↑B or a dot to signify this condition.

At times, the system becomes very busy, and an SI command takes a long time to execute. If a user attempts to enter another command at this time, his typing is ignored and his terminal bell rings to warn him to wait.

10.5 UTILITY COMMANDS

The Monitor provides a number of special purpose commands to aid in program development and use. The Monitor utility commands are summarized in Table 10-2.

TABLE 10-2

MONITOR UTILITY COMMANDS

Command	Explanation
BREAK	Print the current value of the user's delimiter mask.
BREAK C1	Set the user's delimiter mask to C1. (The use of the delimiter mask is discussed in the chapter on assembly language programming).
DUPLEX	Place the user's terminal in duplex mode. All characters typed at the keyboard are automatically printed as they are entered.
RESTART	Print the user's restart address.
RESTART C1	Set the user program restart address to C1. If CTRL/C is typed at the keyboard, Monitor forces a jump to location C1 in the user's program.
SWITCH	Print the current value of the user's switch register.
SWITCH C1	Set the user's switch register to C1. Monitor maintains a switch register for each user. When his program executes on OSR (OR the switch register into the AC) this value is the one which is loaded.
UNDUPLEX	Take the user's terminal out of duplex mode. Input characters are received by the Monitor and by the user program without their being printed at the console.
USER	Print the user's job number, account number, and console number.
USER C1	Print the job number, account number, and console number associated with job C1.
VERSION	Print the version of the Monitor being used.

CHAPTER 11

WRITING ASSEMBLY LANGUAGE PROGRAMS

11.1 INTRODUCTION

In addition to the higher-level programming languages available in the EduSystem 50 library, the user can also code and run programs written in the PDP-8 assembly language, PAL-D (Program Assembly Language). These programs are prepared with EDIT, assembled with PAL-D, then loaded with LOADER.

A user can program EduSystem 50 just as he would any other 4K PDP-8. (Assembly language programs must fit in 4K of core.) All memory reference instructions (AND, TAD, ISZ, DCA, JMS, and JMP) function as on a stand-alone PDP-8. All operate instructions (instruction code 7) also function as on a regular PDP-8 (except that microcoding HLT or OSR with any other operate instruction but CLA gives unpredictable results).

The major difference between EduSystem 50 programming and regular PDP-8 programming is in the IOT (input/output transfer) instructions. Some instructions which are valid on stand-alone PDP-8s, such as CDF, CIF, ION, IOF are considered illegal instructions under timesharing. There are a great many new IOTs within EduSystem 50 that are not valid on a regular PDP-8. Finally, there are IOTs which operate on EduSystem 50 in the same manner as on stand-along PDP-8s. (Table 7-2 is a summary of EduSystem 50 IOT Instructions.)

The way EduSystem 50 actually executes an IOT instruction is also different. Non-IOT instructions (except HLT and OSR) are executed by the hardware, while IOTs (and HLT and OSR) are executed by the EduSystem 50 Monitor.

In general, EduSystem 50 provides the programming capabilities of a 4K PDP-8 and allows programs of considerably greater complexity to be run within the constraints of each user's 4K of core. System Library Programs, all of which were written in assembly language and make use of the EduSystem 50 IOTs dealt with below, are examples of programs which can be run on EduSystem 50.

NOTE: Some of the following instructions do not operate the same under account 1. For information on these instructions, see the Manager's guide.

11.2 CONSOLE I/O

User programs handle console (terminal) I/O in almost the same way as stand-alone PDP-8 programs. The KRB instruction is used to input a character, the TLS instruction to output a character. The KSF and TSF (followed by JMP .-1) can be used but are not needed. Monitor handles all timing problems whether these skip IOTs are present or not.

EduSystem 50 differs from the stand-alone PDP-8 in that under EduSystem 50 the user program interacts with multi-character input and output buffers (maintained by Monitor) rather than with single character registers. Depending on the state of the system, these buffers may have one, many, or no characters in them. During normal program execution, this fact is of no consequence. User programs still send and receive characters one at a time. There are times, however, when it is useful to clear out any and all characters in the buffers; a special IOT exists for this purpose (SBC).

On a stand-alone system, characters are input as soon as they are typed, whether they are of immediate interest or not. Usually, these characters are stored by the program until a terminating (or delimiting) character is found. At this time, the whole line of characters is processed. On a swapping, time-sharing system such as EduSystem 50, this mode of operation is wasteful. It is far more efficient to allow input characters to accumulate in the Monitor input buffer until a delimiter is found. There is an IOT to specify which characters are to be considered delimiters (KSB).

EduSystem 50 also allows programs to input and output strings of characters. The read string (KSR) and send string (SAS) instructions provide a convenient and efficient means of doing lengthy transfers.

All keyboard input uses full-duplexed hardware; there is no wired connection between the keyboard and printer (i.e., characters are not printed on the console as typed). Input characters are echoed to the console under program control rather than by hardware. Because input characters are allowed to accumulate in buffers before being passed to the user program, it is important to have Monitor perform the echoing rather than user programs. There is an IOT (DUP) to set up this automatic echoing as well as an IOT (UND) to inhibit echoing for such operations as reading tapes.

Read Keyboard Buffer (KRB) Octal Code: 6036

Operation: Read the next input character into bits 4-11 of the AC.

Load Teleprinter Sequence (TLS) Octal Code: 6046

Operation: The ASCII character in AC bits 4-11 is printed on the user's console.

Skip on Keyboard Flag (KSF) Octal Code: 6031

Operation: The next instruction is skipped if there is a delimiter character in the user's input buffer.

Read Keyboard String (KSR) Octal Code: 6030

Operation: Execution of this instruction initiates a transfer of one or more characters from the user's keyboard to a designated core area. Before executing KSR, load the AC with the address of a two-word block, where:

Word 1: negative of the number of characters to be transferred.

Word 2: address of the core area into which characters are to be placed minus one.

The transfer is terminated when either:

a. the indicated number of characters have been input or
b. a delimiter is seen. At the end of the transfer, the word count and core address are updated and the AC is cleared.

Send A String (SAS) Octal Code: 6040

Operation: Before executing an SAS load the AC with the address of a two-word block, where:

Word 1: contains the negative of the number of characters to be sent.

Word 2: contains the address - 1 of the first word of the string.

The characters are stored one per word right justified starting at the address specified by word 2. Upon execution of SAS, the system takes only as many characters as will fit in the output buffer. It then makes the appropriate adjustment to word 2 to indicate a new starting address and to word 1 to indicate the reduced character count; it returns to the instruction following the SAS. If the character count is reduced to zero, the instruction following SAS is skipped. The instruction following the SAS usually contains a JMP .-2 to continue the block transfer of terminal characters. The AC is cleared by SAS.

Set Keyboard Break (KSB) Octal Code: 6400

Operation: Rather than activate a user's program to receive each character as it is typed, EduSystem 50 accumulates input characters until a certain character, or characters, is seen. To tell the Monitor which characters to look for (these characters are referred to as delimiters), load the AC with a 12-bit mask before executing a KSB. For each bit in the mask which is set, Monitor considers the corresponding character or characters to be delimiters.

Bit	Specifies
0	0 = check rest of mask
	1 = any character is break
1	301-332 (all letters)
2	260-271 (all numbers)
3	211 (Horizontal tab)
4	212-215 (line feed, vertical tab, form feed, RETURN)
5	241-273 (!"#$%&'()*+,-./:;)
6	240 (space)
7	274-300 (< + > ?@)
8	333-337 ([\]↑←)
9	377 (RUBOUT)
10	375 (ALT MODE)
11	any characters not mentioned above

Duplex (DUP) Octal Code: 6402

Operation: DUP informs Monitor that the user wishes each character typed at the console to be echoed on that console's printer as it is received by Monitor. The DUP instruction does not affect the user's registers.

Unduplex (UND) Octal Code: 6403

Operation: UND informs Monitor that the user wishes to suppress character echoing. This can be done for reasons of privacy or because a program does its own character echoing. The user's registers are unaffected by UND.

Set Buffer Control (SBC) Octal Code: 6401

Operation: SBC permits the user program to clear its terminal input and/or output buffer. Before executing SBC set bits 0 and 1 of the AC as indicated below:

Bit 0 Clear output buffer.
Bit 1 Clear input buffer.

11.3 FILES AND DISK I/O

All user programs can gain access to disk storage. The time-sharing Monitor maintains a pool of available disk space which is allocated in units referred to as segments. Segments are 256 word each. These segments are used to make up user files on the disk. Monitor also maintains, for each user, a directory of all files which he has defined.

The IOTs which allow the user to access the disk are of two types: those which define files on the disk and those which transfer data between a defined file and the user's core.

NOTE

> CREATE and OPEN require that a user set up a file name in core. FINF and WHO return file names to core. Each must be specified in internal code (excess 40 code) as shown in Table 11-1. Characters are packed two to a word.

The first step in defining a file is to create it. Creating a file reserves a single segment of disk storage and associates it with a name. This file can then be extended to any length desired. Extending a file appends more segments to it. Similarly, a file can be reduced by any number of segments. Reducing a file removes the last segment or segments from the file. Reducing a file to zero segments deletes it entirely. Once created, a file can be protected, thereby restricting access to it. When created, a file can be read by any user, but only the creator can write in it. This protection can be reset if desired. Finally, it is possible to rename an existing file.

TABLE 11-1

EDUSYSTEM 50 INTERNAL CHARACTER SET

Character	6-Bit[1] Octal	8-Bit Octal	Character	6-Bit[1] Octal	8-Bit Octal
Space	00	240	@	40	300
!	01	241	A	41	301
"	02	242	B	42	302
#	03	243	C	43	303
$	04	244	D	44	304
%	05	245	E	45	305
&	06	246	F	46	306
'	07	247	G	47	307
(	10	250	H	50	310
)	11	251	I	51	311
*	12	252	J	52	312
+	13	253	K	53	313
,	14	254	L	54	314
-	15	255	M	55	315
.	16	256	N	56	317
/	17	257	O	57	317
0	20	260	P	60	320
1	21	261	Q	61	321
2	22	262	R	62	322
3	23	263	S	63	323
4	24	264	T	64	324
5	25	265	U	65	325
6	26	266	V	66	326
7	27	267	W	67	327
8	30	270	X	70	330
9	31	271	Y	71	331
:	32	272	Z	72	332
;	33	273	[	73	333
<	34	274	\	74	334
=	35	275	]	75	335
>	36	276	↑	76	336
?	37	277	←	77	337

[1]The 6-bit octal code is used to store passwords and file names only.

None of these actions affect the contents of the file-they only reserve space on the disk. Until it has been written in, the actual content of a file is unspecified. Extending a file does not alter the content of the file as it previously existed. Once defined, files can be used to read and write data. Any number of words (1 to 4096) can be moved from any part of the user's core to any part of a file (subject to file protection). The user program specifies a location in core and a word count. This indicates how many words are to be transferred and from (or to) where in core they are to be moved. Also specified is a disk file address indicating what part of the file is involved. This address is the address of a word in the file. Files are addressed in the same manner as core: in 12-bit words. Unlike core, however, files can be longer than 4K. To address these files provision is made for a 24-bit disk file address, containing the high-order and low-order file addresses.

File addresses are independent of any consideration of segments. The file address is meaningful only in defining files. Files can be read and written across segment boundaries without restriction. (The user cannot read or write beyond the last segment boundary.)

When it executes a file read or write IOT, the system updates the core address and word count and places an error code in the error word (see RFILE) if any error is detected. At the end of a successful transfer, the word count is set to zero and the core address set to the last word transferred. If the transfer cannot be completed for some reason, the word count and core address indicate how much of the transfer was successful; the error word indicates the cause of the failure. All file operations except CREATE (and OPEN) require that the file be open. Up to four files can be open at a time. The process of opening a file associates it with one of four internal file numbers (0, 1, 2 or 3). All file IOTs except CREATE and OPEN, are specified in terms of one of these internal file numbers, rather than a file name. IOTs operate on the file which is indicated by

that internal file number at the time. It is therefore possible to write file handling programs which are independent of the actual file(s) they operate on.

Each user has a disk quota, defined by the system manager. When a file is extended beyond this quota, the Monitor prints the message [EXCEEDING DISK QUOTA]. When this happens, the user may no longer create files and may only extend. The amount by which a user may exceed his disk quota is called the "grace" quota, and is defined by the system manager.

File IOTs, that are successfully completed, return with the AC cleared. If an error was found which prohibited execution of the IOT, one of the following error codes is returned.

Code	
4000	There was no file opened on the specified internal file number.
4400	Attempting to redefine a file which is open to another user.
5000	Attempting to create a file for a user whose directory is full, or who has exceeded his disk quota.
5400	Bad directory.
6000	File protection violation.
6400	Invalid file name.
7000	Attempting to open a nonexistent file.
7400	Disk is full.

Create a File (CRF) Octal Code: 6610

Operation: The user can request the system to create a new file of one segment. The user program provides the new name for the file.

Load the AC with the beginning address of a 3-word block, where:

Words 1 through 3: contain the 6-character name.

If there is some reason why the request cannot be granted, the system will return a non-zero error code in the AC. The protection code of a newly created file is 12.

Extend A File (EXT) Octal Code: 6611

Operation: To extend the length of an existing file, that file must be currently open. Load the AC with the beginning address of a 2-word block, where:

Word 1: contains the internal file number of the file to be extended.

Word 2: contains the number of segments the system should append to the file.

If for some reason the request to extend a file cannot be granted, the AC will contain 4000, 6000, or the number of segments it failed to append.

Reduce A File (RED) Octal Code: 6612

Operation: To reduce the length of an existing file, that file must be currently open. Load the AC with the beginning address of a 2-word block, where:

Word 1: contains the internal file number of the file to be reduced.

Word 2: contains the number of segments to be removed.

This request is granted unless the file to be reduced is currently opened to another user or if the file is write protected against the user.

Rename A File (REN) Octal Code: 6600

Operation: REN is used to change the name of a file. Load the AC with the address of a 4-word block where:

Word 1: contains the internal file number associated with the file whose name is to be changed.

Words 2-4: contains the new name. This name is in 6-bit characters packed two in a word.

Protect A File (PROT) Octal Code: 6604

Operation: The owner of a file can protect his file from unauthorized attempts to access it by using this instruction. Before executing PROT, load the AC with:

Bits Ø to 4	File extension code (for further information on extensions see the description of the PROTECT Monitor command).
Bits 5 and 6	Internal file number of the reserved file to be protected.
Bit 7	Write protect against owner.
Bit 8	Write protect against users whose project number is same as owner's.
Bit 9	Read protect against users whose project number is same as owner's.
Bit 10	Write protect against users whose project number differs from owner's.
Bit 11	Read protect against users whose project number differs from owner's.

A file must be opened before it can be protected. PROT is legal only when performed by the file owner, i.e., the user who created the file. All attempts to access the file which violate any of the protection flags are considered illegal. (For further information on project numbers, see Appendix C).

Open A File (OPEN) Octal Code: 6601

Operation: OPEN is used to associate a file with an internal file number, which is necessary because all file operations are in terms of the internal file numbers. Before executing the OPEN IOT, load the AC with the beginning address of a 5-word block, where:

Word 1:	contains the internal file number.
Word 2:	contains the account number of the owner of the file. If 0, the account number of the current user is specified.

Word 3-5: contain the name of the file to be opened. This name is in 6-bit characters packed two to a word.

If there was another file associated with the internal file number before the execution of the OPEN IOT, it is closed automatically before the new file is associated with the internal file number.

Close A File (CLOS) Octal Code: 6602

Operation: CLOS terminates the association between files and their internal file numbers. Before executing CLOS, load the AC with a selection pattern for the internal file numbers whose associated files are to be closed. The file is closed if bit I is 1, where I = bit 0, 1, 2, or 3.

READ File (RFILE) and Write File (WFILE) Octal Code: 6603 & 6605

Operation: Once the association of a file with an internal file number has been made, these IOTs allow the actual file reference to be made. They are illegal on a file that has not been opened (associated with an internal file number).

To read or write a file, load the AC with the address of a 6-word block, then execute the IOT. The format for the 6-word block is:

Word 1: contains the high-order file word address.
Word 2: contains the internal file number
Word 3: contains the negative of the number of words for the operation. This number is either the number of words to be read or the number of words to be written.
Word 4: contains a pointer to the beginning address - 1 of a buffer located in the user program. On a read operation this buffer receives the information from the file: on a write operation this buffer holds the information that is to be sent to the file.

Word 5: contains the least significant 12 bits of the initial file word address to begin the operation.

Word 6: contains an error code:

0 if no error
1 if parity error
2 if file shorter than word count
3 if file not open
4 if protection violated

The read or write begins at the word specified by words 1 and 5. For example:

```
      TAD X
      WFILE
      :
X,    .+1
      Ø
      1
      -2ØØ
      6477
      2ØØ
```

means: write 200 (octal) words starting at word 200 of the file that is associated with internal file number one from a core area starting at location 6500.

After completion of the transfer, the word count (word 3) and core address (word 4) are updated. If an error was detected the appropriate error code is placed in word 6.

File Information (FINF) Octal Code: 6613

Operation: FINF enables a user program to determine what file, if any, is associated with an internal file number. Load the AC with the beginning address of a 7-word block, where:

Word 1: contains the internal file number for which the user program wishes information.

Words 2 through 7: contain the information that the system returns after executing FINF.

Word 2: contains the account number of the owner or zero if no file is associated with the internal file number, that is, the file is not open.

Words 3-5: contains the name of the file in 6-bit code.

Word 6: contains the file extension code and protection code. See Monitor PROTECT command for more details.

Word 7: contains the number of segments which compose the file.

11.4 ASSIGNABLE DEVICES

Users can access both their own terminal and the disk; with the remaining system devices (referred to as the assignable devices) this is not true. One function of the Monitor is to ensure that device usage never conflicts. Only one user at a time can access the high-speed paper-tape reader or punch, or any one of the DECtapes. To ensure that only one user can access a device, EduSystem 50 requires that the device be assigned before it is used. After a device is assigned, it is not available until it is released by its owner.

Once assigned, the device is programmed much as on a stand-alone PDP-8. The RRB instruction is used to read a character from the high-speed reader; the PLS instruction is used to punch one on the high-speed punch. The skip IOTs (RFS and PSF) can be used (followed by JMP.-1) but are not necessary. For block transfers, there are two string transfer commands: RRS and PST.

The DECtape instructions have been simplified. A single instruction, DTXA, initiates the transfer of a block of data. The DTRB instruction is then used to determine if the transfer was successful. The skip instruction, DTSF, can be used (followed by JMP.-1) but is not necessary.

Executing any of the assingable device IOTs without first assigning the device gives the following results: (a) If the device is assigned to another user, the instruction is considered illegal; program execution is now terminated and an error message printed; (b) If the device is available it is automatically assigned before execution of the IOT. The device then belongs to this user until he releases it.

Because these devices are shared by all users, the Monitor must ensure that they are operable at all times. In particular, the Monitor must ensure that a user is not waiting for a device which is not available. This situation can arise when trying to use the punch when it is turned off, or when the reader has read off the end of a tape. All these conditions, known as "hung devices" are considered to be system errors. If the program doing the transfer has been enabled for system errors (by executing an SEA), control transfers to the error routine indicated which must clear the error flag in the status word before continuing (See Section 11.6). If the user program has not been enabled for system errors, a hung device causes the program to be terminated and an error message is printed.

When the paper tape punch or line printer is off line or hung, the Monitor takes special action so that it is usually possible to continue with little or no data loss. When these devices are hung, the output buffer is not cleared. A system error is generated, and regenerated every few seconds until the condition is cleared. If the user program has not enabled errors, the result will be a printed "HUNG DEVICE" message, and then the terminal bell will ring, trying

to persuade the user to do something. There are only two things the user may do to remove himself from this condition. If he is not interested in continuing, he may release the hung device. If he does wish to continue he should put the device on line, and it should take off. He may now type "START" to continue program execution.

Assign Device (ASD) OCtal COde: 6440

Operation: If the device specified by the content of the AC is available, it is assigned to the users program and the AC is cleared. Otherwise, the number of the job owning the device is placed in the AC. If the device does not exist, 7777 is returned in the AC.

4000	Paper-tape reader
4001	Paper-tape punch
4003	Line printer
4004	Card reader
4005+N	DECtape unit N, N=0-7
4015	RK8E drive N, N=0-3

The assignment is in effect until a corresponding REL instruction or LOGOUT is executed.

Release Device (REL) Octal Code: 6442

Operation: The device specified by the contents of the AC is releaseed (providing it was owned by the user executing the REL). The AC is cleared. Releas ing a device makes it available to other users.

Skip on Reader Flag (RSF) Octal Code: 6011

Operation: The contents of the PC are incremented by one so that the next sequential instruction is skipped.

Read Reader Buffer (RRB) Octal Codes: 6012 & 6016

Operation: The contents of the reader buffer are transferred into bits 4 - 11 of the AC. This instruction does not clear the AC. If the reader buffer is empty, the user program is put into a wait state until the buffer is full or an end-of-tape condition is detected.

Reader Clear Buffer (RCB) Octal Code: 6017

Operation: Any characters, which may have been read from the high-speed reader but not passed to the user, are cleared fro the buffer.

Read Reader String (RRS) Octal Code: 6010

Operation: This instruction initiates a transfer from the high-speed reader to a selected area in the users core. Before executing RRS, load the AC with the address of a 2-word block where:

word 1: minus the number of characters to be transferred.

word 2: the address of the user core area minus one.

The transfer is terminated by either of two conditions: (a) the word count (word 1) is zero indicating that the required number of characters have been read or (b) the reader has read off the end of the tape (a system error condition). In either case, the word count and core address are updated. RRS clears the AC.

Load Punch Buffer Sequence (PLS) Octal Code: 6026

Operation: The ASCII character in AC bits 4 - 11 is transmitted to the high-speed punch. PLS does not clear the accumulator.

Skip on Punch Flag (PSF) Octal Code: 6021

Operation: The contents of the PC are incremented by one so that the next sequential instruction is skipped.

Punch String (PST) OCtal Code: 6020

Operation: PST allows a user program to punch a string of charactses. Before executing PST, load the AC with the beginning address of a 2-word block where:

Word 1: contains the negative of the number of characters to be punched.

Word 2: contains the beginning address - 1 of the string to punched; the characters should be right justified one per word.

After execution of PST, the system takes only as many characters as fit in the punch buffer; it then makes the appropriate adjustment to word 2 to indicate a new starting address and to word 1 to indicate the reduced character count. It returns to the instruction following the PST which may be a JMP .-2 to continue the transfer. If the character count is reduced to zero, the instruction following PST is skipped. The AC is cleared by PST.

Line Printer Print (LPC) Octal Code: 6666

Operation: The ASCII character in bits 4 through 11 of the AC are placed into the line-printer buffer to be printed. LPC does not change the AC.

Line Printer Skip on Flag (LSF) Octal Code: 6661

Operation: The contents of the PC are incremented by one so that the next sequential instruction is skipped.

Line Printer Send-A-String (LST) Octal Code: 6660

Operation: LST allows the user program to print a string of characters. Before executing LST, load the AC with the beginning address of a 2-word block, where:

Word 1: Contains the negative of the number of characters to be printed.

Word 2: Contains the beginning address -1 of the string to be printed; the characters should be right justified one per word.

After execution of LST, the system takes only as many characters as fit in the punch buffer; it then makes the appropriate adjustment to word Z to indicate a new starting address and to word 2 to indicate

the reduced character count. It returns to the instruction following the PST which may be a JMP .-2 to continue the transfer. If the character count is reduced to zero, the instruction following PST is skipped. The AC is cleared by LST.

Load Status Register A (DTXA) Octal Code: 6764

Operation: DTXA allows a user program to read and write records (129-word blocks) on DECtape. Load the AC with the beginning address of a 3-word block, where:

Word 1: contains:

Bit 1	Meaning
0-2	contains the transport unit select number,
3	is set to 1 to read/write in reverse
4-5	should be Ø
6-8=2	for read data function,
=4	"for write data function"
9-11	should be Ø

Word 2: contains the DECtape block number.

Word 3: contains the beginning core address - 1 of a 2Ø1 (octal) word buffer.

After DTXA is given, the DECtape request is placed in the DECtape request queue. Control does not return to the user until the request has been honored or an error has occurred. DTXA does not update word 3. The AC is cleared by DTXA.

Skip on Flags (DTSF) Octal Code: 6771

Operation: The contents of the PC are incremented by one so that the next sequential instruction is skipped.

Read Status Register B (DTRB) Octal Code: 6772

Operation: The content of DECtape status register B is loaded into the AC. This instruction is also used with the card reader and RKØ5. See Read Status (RDS) for details.

Read card Alphanumeric RCRA) Octal Code: 6632

Read card Binary (RCRB) Octal Code: 6634

Read card Compressed (RCRC) Octal Code: 6636

Before executing the above instructions, load the AC with the address minus 2 of an 80 word buffer. A card is read and the data is put into the buffer in the same form as the corresponding hardware IOT. The UUO returns in the AC the number of characters successfully transferred to the user's buffer. (See also 6772 - RDS.)

Disk Load Address and go (DLAG) Octal Code: 6743

Allows the user to read or write on the RK8E. To use, load the AC with the address of a three word block, where

Word 1: Bit 0 = 0 for a read,
= 1 for a write
Bits 3-8 contain the number of pages to read/write,
1 to 40.
Bits 9-10 contain the drive number 0 to 3.
Bit 11 contains the high order sector address.

Word 2: contains the core buffer address minus one,

Word 3: contains the low order sector address.

Upon return, the AC contains the number of blocks transferred. To determine error conditions, see 6772-RDS.

The disk transfer is made in 400 (octal) word blocks. Each RK05 drive contains 14540 (octal) blocks. To specify the initial block number, the high order bit goes into word 1 of the parameter block, and the remaining 11 bits into word 3. If the transfer requests an odd number of pages on a write, the last page of the last block on the disk will contain zeros. Upon return, the AC contains (P+1)/2, where P is the number of pages successfully transferred.

Read Status (RDS) Octal Code: 6772

The contents of the device status register are placed into the AC.

The information obtained pertains to the RK8E, DECtape, or Card Reader, depending on which was most recently used. The contents of the device status register are:

RK8E: RK8E Status Register

Bit	Assignment
0:	Control done
1:	Heads in motion
3:	Seek fail
4:	File not ready
5:	Busy error
6:	Time out error
7:	Write lock out error
8:	CRC error
9:	Data request late
10:	Drive status error
11:	Cylinder address error

DECtape: TC08/TC01 Status register B

Bit 0: Error flag

1: Mark track error

2: End of tape

3: Select error

4: Parity error

5: Timing error

11: DECtape flag (normal)

The status register for DECtape may also include 4ØØØ or 4ØØ1. These are software generated errors such as block number out of range.

Card reader: The device status register contains the address of the last word of data transferred to the user's buffer. In addition, the device status retister may contain 7777. This indicates that CTRL/B followed by S was typed while a DECtape or RKØ5 transfer was in progress, and the transfer was not finished.

11.5 PROGRAM CONTROL

There are a number of ways that the status of a running program can be changed. The program can be terminated in one of three ways: by execution of a HLT, by the user typing CTRL/B followed by S to force a program halt, or by a program error which forces Monitor to terminate the program after printing an error message.

It is also possible for the status of a running program to change without it being terminated. First, the user program can request that it handle its own program error conditions. In this case, Monitor does not terminate a job on an error; instead, it transfers control to a user error handler. This error handler then determines what the error was, by a CKS instruction and takes appropriate action. Monitor also provides the program with an interrupt key, CTRL/C. If the user types a CTRL/C, the Monitor unconditionally transfers control to a restart address. Thus, the user program can handle its own restarts.

Halt (HLT) Octal Code: 7402

Operation: This instruction is used to stop the user program and return control to Monitor. Executing HLT is equivalent to typing ↑BS followed by RETURN.

Set Restart Address (SRA) Octal Code: 6417

Operation: This instruction allows the user to specify an address to which control is transferred when an ↑C is typed on the user's console. Load the AC with the restart address and execute SRA. If ↑C is detected, the program's input and output buffers are cleared, the AC and Link are cleared and control goes to the restart address.

Set Error Address (SEA) Octal Code: 6431

Operation: This instruction allows the user to specify an address to which control is transferred in the event of a system error. Load the AC with an address before executing SEA. If a system error is detected, Monitor simulates a JMS to the error address. The program counter is stored in the error address and control transferred to the error address +1. AC, Link, and input/output buffers are not affected. The error code of the system error is in STR0 bits 9-11. The error routine must read these bits (by a CKS) to determine the cause of the error, then clear them by means of a CLS.

The only error code that occurs, for example, in the course of normal system usage is due to a hung device. This error occurs when the user attempts to use a punch, line printer, or reader which is not turned on, or allows the paper-tape reader to run off the end of a tape. In the case of the line printer or the punch, the error routine must release the device or the device must be turned on to clear the error condition. The illegal IOT error probably means that an assignable device IOT was executed without the device first being assigned. Swap and file errors occur if a hardware error is detected while Monitor is swapping user programs or while reading or writing

file directories. These are system malfunctions from which there is no recovery.

11.6 PROGRAM AND SYSTEM STATUS

Because EduSystem 50 programs run under control of a time-sharing Monitor, it is important for them to determine their status within the system and the status of the system as a whole. Several IOTs, listed below, have been defined for this purpose.

Check Status (CKS) — Octal Code: 6200

Monitor maintains for each user a complete set of status information, his program's running status and the state of his input/output devices. This status information, stored in three words, can be accessed by a running program with the CKS instruction. Before executing a CKS, load the accumulator (AC) with the address of a three-word block. Executing CKS stores the three status words (STR0, STR1, and DEVICE STATUS REGISTER) in the three-work block and clears the AC.
The formats of these registers are:

STR0 Bits

0	Run Bit	User program is in the run state
1	Error Enable	Program handles its own errors
2	JCOMBD	Program was compute bound
3	JSPEEK	User has R privilege
4	JSACC	User is privileged account
5	JSIOT	System use only
6	JSIOTC	System use only
7	Not used	Not used
8	JSINER	System use only
9-11	Error Code	System detected error condition 1 Illegal IOT 2 Swap read error 3 Swap write error 5 Disk file error 6 Hung device

STR1 Bits

0	Timer	Time is up
1	File 0	Internal file 0 is not busy
2	File 1	Internal file 1 is not busy
3	File 2	Internal file 2 is not busy
4	File 3	Internal file 3 is not busy
5	Delimiter	There is a delimiter in the input buffer
6	Line Printer	Output buffer is not full
7	Teleprinter	Output buffer is not full
8	Reader	Character in reader buffer
9	Punch	Punch buffer is not full
10	Error	System error has occurred.
11	Wait	Job is not waiting

Device Status Register: The contents of the Device Status Register pertain to the card reader, DECtape, or RKØ5 as described under the Read Status (RDS) IOT.

OR With Switch Register (OSR) Octal Code: 7404

Operation: The content of the user's switch register is inclusively ORed into the AC.

Set Switch Register (SSW) Octal Code: 6430

Operation: The content of the AC is stored in the user's switch register. The AC is cleared.

Assembly language programs run under control of the Monitor. The following IOTs are defined to allow a program to determine the status of the system as a whole.

Segment Size (SIZE) Octal Code: 6614

Operation: The segment is the basic unit of on-line file storage. The size of a segment (0400 octal) is returned in the AC.

Segment Count (SEGS) Octal Code: 6406

Operation: The number of available disk segments is returned in the AC.

Account (ACT) Octal Code: 6617

Operation: The account number (of the job number in the AC) is returned in the AC. If AC is 0, the account number for the current job is returned. If the requested job does not exist, zero is returned.

Who (WHO) Octal Code: 6616

Operation: The account number and password of the current job are returned to the 3-word block whose address is in the AC and the AC is cleared.

User (USE) Octal Code: 6421

Operation: Return in the AC the number of the current job.

Console (CON) Octal Code: 6422

Operation: Return in the AC the console unit number assigned to the job whose number is in the AC, if that console number does not exist, -1 is returned.

User Run Time (URT) Octal Code: 6411

Operation: Load the AC with the address of a 3-word block, where word 1 contains the number of the job for which the run time is sought. The run time is returned in the last two locations of the block. If job 0 is specified, the run time of the current job is returned. The AC is cleared.

Time-of-Day (TOD) Octal Code: 6412

Operation: Returns the time of day in the two locations starting at the location of the address in the AC. The time of day is returned in clock ticks since midnight. The AC is cleared. The clock ticks 10 times per second.

RETURN CLOCK Rate (RCR) Octal Code: 6413

Operation: The number of clock ticks per second is returned in the AC. The clock ticks every 100 ms, the AC will be set to 12 (octal).

Date (DATE) Octal Code: 6414

Operation: Returns the date in the AC. The format of this 12-bit number is:

```
DATE=((YEAR-1974)*12+(MONTH-1))*31+DAY-1
```

Skip on EduSystem 50 (TSS) Octal Code: 6420

Operation: This instruction is used by programs which run under both EduSystem 50 and on a standard PDP-8. Under EduSystem 50, the instruction following TSS is skipped and the Monitor version number is returned in the AC. On a standard PDP-8, the IOT has the effect of a NOP instruction.

Quantum Synchronization (SYN) Octal Code: 6415

Operation: Upon execution of this instruction, the system dismisses the user program and sets it in the run state so that it will be run again in turn. Ordinarily, this instruction is used to ensure a full time quantum to perform some critical operation.

Set Time (STM) Octal Code: 6416

Operation: The system provides a clock time for each user program. By means of this IOT, the time can be set to "fire" after a specified number of clock ticks have elapsed. Load the AC with the time (in seconds) to prime the timer. Upon execution of the STM instruction, the system sets the time to "fire" in the specified number of seconds, clears the AC, and dismisses the job. After the specified time has elapsed, the job is restarted. Due to the time-sharing environment, this time may vary by a second or two.

11.6 1/2 SPECIAL INSTRUCTIONS

The following instructions are usable only on certain TSS/8 systems, and are not included in the PALD symbol table.

Grt Flags (GTF) Octal Code: 6004

Operation: Upon execution of this instruction, the link is placed into AC bit 0, and the EAE GT flag (if present) is placed into AC bit 1. The rest of the AC is cleared. This instruction is valid only on a PDP-8/E.

Restore Flags(RTF) Octal Code: 6005

Operation: Upon execution of this instruction, AC bit 0 is placed into the link and AC bit 1 is used to set or clear the EAE GT flag (if present). The AC is not changed by RTF. This instuction is valid only on a PDP-8/E.

Skip or Greater Than Flag(SGT) Octal Code: 6006

Operation: If the system includes an EAE, and if its GT flag is set, this instruction will cause the PC to be incremented by one so that the next core location is skipped. This instruction is valid only on a PDP-8/E.

In addition, all operate type instructions which the computer is capable of executing may be used. This includes the BSW and MQ instructions on any PDP-8/E, and all the EAE instructions on systems which have an EAE.

11.7 PDP-8 COMPATABILITY

Programming EduSystem 50 in assembly language is very similar to programming a stand-alone PDP-8. All instructions except the IOTs operate identically in either case. As discussed previously, programming such devices as the terminal and high-speed reader/punch for EduSystem 50 is somewhat simpler. EduSystem 50 runs programs which include timing loops. Thus, programs written for stand-alone PDP-8s with terminal and high-speed reader or punch will run on EduSystem 50, although generally not as efficiently as programs which are written specifically for EduSystem 50.

The same is not true for disk and DECtape operations because EduSystem 50 uses a simplified programming structure for these devices. The actual differences in coding are very small. It is a simple task to adapt previously written code for EduSystem 50 disk and DECtape.

There are a few standard changes which users generally make in adapting PDP-8 code to EduSystem 50. Monitor does the echoing rather than the user program. The TLS which does the echo can be deleted and a DUP instruction added somewhere near the start of the program. Also, for efficiency, the EduSystem 50 delimiter capability can be used. A KSB in the program determines what the delimiters are.

Many PDP-8 programs execute a reader and punch IOT early in the program, to initialize the device, whether they are actually to be used or not. If the devices are free, they are assigned and thus made unavailable to other users. If they are unavailable, the program terminates on an illegal IOT. Thus, it is important not to execute these IOTs randomly. If a disk card reader or DECtape is involved,

the actual transfer code must be altered to conform to EduSystem 50. (The fact that core page 37, locations 7600 through 7777, is available to EduSystem 50 programs is useful in making these changes. New code can be placed in the area normally reserved for the Binary Loader.)

The most difficult code to convert is that code which operates under interrupt. To be run under EduSystem 50, these programs must be recoded so as not to use the interrupt.

IOTs for nonexistent devices are ignored as are CDFs and CIFs. It must be remembered that many IOTs have been redefined for use as special EduSystem 50 instructions. In all other situations, EduSystem 50 remains compatible with stand-alone systems whenever possible.

APPENDIX A

EDUSYSTEM 50
MONITOR COMMAND SUMMARY

A.1 MONITOR COMMANDS

A Monitor command is a string of characters terminated by a semicolon (;), a colon (:), or a carriage return (RETURN key). Parameters of commands can be octal numbers, decimal numbers, character strings, or single letters. In the following summary, parameters are coded as follows:

C1, C2,...	represent octal numbers
D1, D2,...	represent decimal numbers
L1, L2,...	represent single letters
S1, S2,...	represent character strings

A.1.1 LOGGING IN AND OUT

KJOB	used as an alternative to LOGOUT
LOGIN C1 S1	Request to login: C1 = user's account number S1 = user's password
LOGOUT	Request to logout: processing and device time are printed.
TIME C1	C1 = job number A) If C1 is omitted and the user is logged in, the processing time of the current job is printed. If C1=0, or if the user is not logged in and C1 is omitted, the time of day is printed.

A.1.2 DEVICE ALLOCATION

ASSIGN D C1	Reserve DECtape unit: C1 = DECtape unit number
ASSIGN K C1	Assigns a specific RKØ5 unit
ASSIGN L1	Reserve device:
	L1 = R for paper tape reader
	P for paper tape punch
	D for any DECtape unit
	L for line printer
	C for Card Reader
	K for any RKØ5 unit
RELEASE D C1	Release DECtape unit: C1 = DECtape unit number
RELEASE K C1	Release an RKØ5 unit: C1 = RKØ5 drive number
RELEASE L1	Release device:
	L1 = R for paper tape reader
	P for paper tape punch
	L for line printer
	C for card reader

A.1.3 FILE HANDLING

CLOSE S1	Close files:
	S1 = list of internal file numbers
CREATE S1	Create new file:
	S1 = name of new file
EXTEND C1 D1	Extend length of file:
	C1 = internal file number
	D1 = number of segments to be added to end of file
F C1	Print information about an open file
	C1 = internal file number

OPEN C1 S1 C2 Establish association between internal file number and file:

C1 = internal file number

S1 = file name

C2 = account number. If omitted, the user's account is assumed.

PROTECT C1 C2 Protect a file:

C1 = internal file number

C2 = new file protection mask (See 10.3.4)

REDUCE C1 D1 Reduce length of file:

C1 = internal file number

D1 = number of segments to be removed from end of file

RENAME C1 S1 Rename a file:

C1 = internal file number

S1 = new name of file

A.1.4 CONTROL OF USER PROGRAMS

DEPOSIT C1 C2...Cn Store in core memory:

C1 = location

C2 = contents to be stored in location C1

C3 = contents to be stored in location C1+1, etc.

EXAMINE C1 D1 List specified contents:

C1 = first location

D1 = number of location to be listed, D1≤10 (decimal)

RESTART Print the program restart address

RESTART C1	Set program restart address.
START	Restart user program.
START C1	Execute user program: C1 = starting location

A.1.5. UTILITY COMMANDS

BREAK	Print keyboard break mask.
BREAK C1	Set keyboard break mask: C1 = new mask
DUPLEX	Echo typed characters on printer.
LOAD C1 S1	Load Core image:
LOAD C1 S1 C2	C1 = owner's account number; if not specified the user's account is assumed.
LOAD C1 S1 C2 C3	C2 = file address of first word to be loaded; if not specified, Ø is assumed.
LOAD S1	C3 = core address of first word to be loaded; if not specified; Ø is assumed.
LOAD S1 C2	C4 = core address of last word to be loaded; if not specified, highest possible value is assumed.
LOAD S1 C2 C3	
LOAD S1 C2 C3 C4	
R S1	Run system file:
R S1 C1	S1 = name of file
	C1 = beginning address; if omitted, Ø is assumed.
RUN S1	Run user file:
RUN C1 S1	S1 = name of file
RUN S1 C2	C1 = owner's account number; if omitted, the user's account is assumed.
RUN C1 S1 C2	C2 = starting address; if omitted, Ø is assumed.
S	Stop execution
	Save Core image:

SAVE C1 S1	C1 = owner's account number; if not specified the user's account number is assumed.
SAVE C1 S1 C2	S1 = name of file
SAVE C1 S1 C2 C3	C2 = file address of first word to be saved; if not specified, Ø is assumed.
SWITCH	Print the current value of the user's switch register.
	Set switch register:
SWITCH C1	C1 = word to be set
TALK C1 S1	Send a message to console C1:
	C1 = destination console
	S1 = message
UNDUPLEX	Inhibit echo of characters typed to a user program
USER	Print the user's job number, account number, and console number.
USER C1	Print the job number, account number, and console number of job C1.
WHERE	Print the current value of the user's PC, AC, link, switch register, and EAE registers.

APPENDIX B

CHARACTER CODES

The ASCII[1] character codes shown in the following table are used by EduSystems as the argument in the CHR$ function. For each ASCII code a second acceptable form is permitted in CHR$. The second code is obtained by adding 128 to the code given in the following table. For example, CHR$ would print A in response to either 65 or 193 as an argument. These codes are also used with the CHANGE statement in EduSystem 50.

Character	ASCII Code No. (Decimal)	Character	ASCII Code No. (Decimal)
linefeed	10		
formfeed	12		
RETURN	13		
space	32	@	64
!	33	A	64
"	34	B	66
#	35	C	67
$	36	D	68
%	37	E	69
&	38	F	70
'	39	G	71
(	40	H	72
)	41	I	73
*	42	J	74
+	43	K	75
,	44	L	76
-	45	M	77
.	46	N	78
/	47	O	79

[1]An abbreviation for American Standard Code for Information Interchange.

Character	ASCII Code No. (Decimal)	Character	ASCII Code No. (Decimal)
0	48	P	80
1	49	Q	81
2	50	R	82
3	51	S	83
4	52	T	84
5	53	U	85
6	54	V	86
7	55	W	87
8	56	X	88
9	57	Y	89
:	58	Z	90
;	59	[	91
<	60	\	92
=	61	]	93
>	62	↑	94
?	63	←	95

APPENDIX C

STORAGE ALLOCATION

C.1 STORAGE MAP

The system's storage allocation is illustrated below.

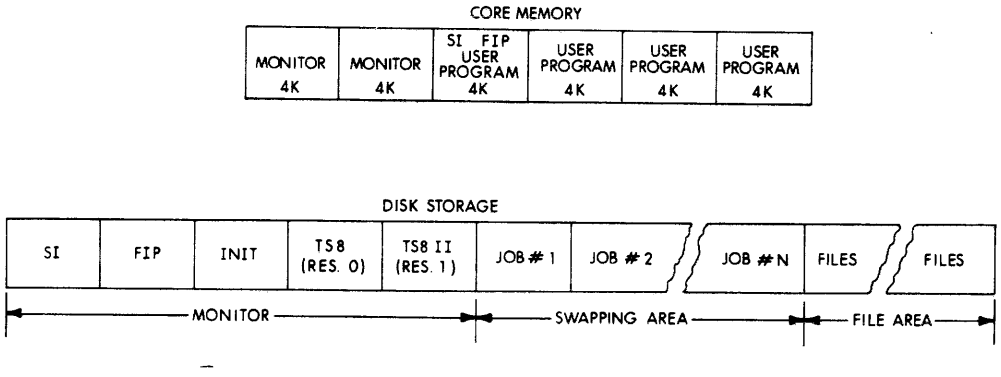


Figure C-1. EduSystem 50 Storage Map

C.2 FILE DIRECTORIES

There are two directories on the disk: the Master File Directory (MFD) referenced mainly by the system, and the User File Directory (UFD), referenced by the user. One of the functions of the MFD is to service the UFD. A UFD is a particular user's file directory containing the names of programs he has created on the disk.

The UFD is a file like any other file except that its filename is the project-programmer number and password. When a user is logged in under a specific number and references the disk, he is actually referencing his own file area on the disk through the UFD which has his project-programmer number as its name.

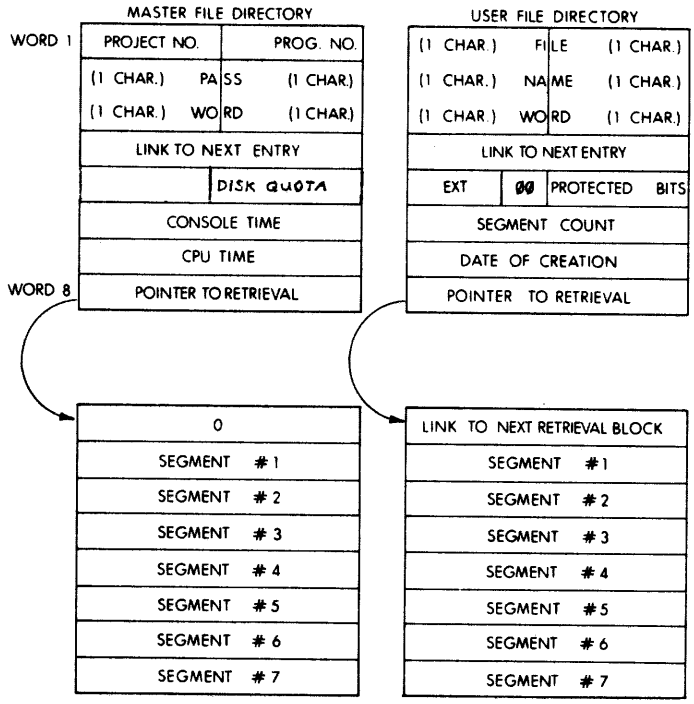


Figure C-2. File Directories

System account numbers are a combination of a project number and a programmer number. If expressed as a four-digit octal number, the first two digits of the account number are the project number, and the last two digits are the programmer number.

APPENDIX D

ERROR MESSAGES

Message	Program	Explanation
ABORT	BASIC	BASIC can not run for some reason. Perhaps the user's disk quota is exceeded.
ARRAY OR RECORD USED BEFORE DEFINITION	BASIC	The RECORD statement must occur before any reference to it is made. A DIM statement must occur before an array is used. (RECORD and DIM are placed at the beginning of a program.)
BAD FILE FORMAT	BASIC	The program specified in response to OLD PROGRAM NAME was not acceptable to BASIC. This is generally caused by: (1) trying to load an obsolete compiled (.BAC) file, or (2) trying to load a non-BASIC (FORTRAN or PAL-D) program.
BAD FILE NAME	BASIC	The file name used is not valid; e.g., it does not begin with a letter.
BAD SLEEP ARGUMENT		The argument of the SLEEP statement must have a number greater than or equal to 0, and less than or equal to 4095.

Message	Program	Explanation
BAD VALUE IN CHANGE STATEMENT		While performing CHANGE A TO A$, one of the elements of the array A was found to contain an illegal value.
CAN'T CREATE FILE		An OPEN statement tried to create a file, but there is: (a) no disk space available, (b) no file name specified, or (c) a null string has been given as the file name.
CAN'T DELETE FILE		UNSAVE cannot delete a file. This is usually due to the fact that another user has the file open, or the file is protected with a code $\geq$ 20.
CAN'T FIND LINE	BASIC	An attempt has been made to edit a nonexistent line.
CAN'T FIND "NAME" IN SYSTEM LIBRARY	BASIC	The requested OLD file cannot be found.

Message	Program	Explanation
CHAIN TO BAD FILE	BASIC	The file specified by the CHAIN has an invalid format; it is not a BASIC format file. The "PROGRAM IS..." message will follow this error message. The program name will be the name of the bad file.
DEF STATEMENT MISSING	BASIC	A function needing a DEF statement exists in the program.
DEVICE BUSY	BASIC	The user tried to OPEN DECtapes 0-7, line printer, or paper tape punch, but the device was unavailable, and there was no ELSE clause in the OPEN statement.
DIMENSION TOO LARGE	BASIC	Too large an array to fit in the available core.
DISK FULL	BASIC	There is no more storage space on the system disk, or the user has exceeded his disk quota.
DUPLICATE FILE NAME	BASIC	An attempt has been made to SAVE a program but one already exists with that name.

Message	Program	Explanation
EXECUTE ONLY	BASIC	An attempt has been made to list, edit, or alter a BASIC compiled program. It may be run only.
FOR WITHOUT NEXT	BASIC	There is an unmatched FOR statement in the program.
GET BEYOND END OF FILE	BASIC	Disk data file is too small to have a record with the number specified in the GET statement at line n.
GET/PUT ERROR	BASIC	A hardware error occurred in GET or PUT. (This is usually due to a DECtape unit being write-locked.)
GOSUB-RETURN ERROR	BASIC	Subroutines are too deeply nested or a RETURN statement exists outside a subroutine.
ILLEGAL CHARACTER	BASIC	The user attempted to use an illegal character in the statement being processed.
ILLEGAL CONSTANT	BASIC	The format of a constant in the statement being processed is not valid.

Message	Program	Explanation
ILLEGAL FORMAT	BASIC	The structure of the statement does not agree with BASIC syntax.
ILLEGAL FOR NESTING	BASIC	FOR NEXT loops are too deeply nested or NEXT appears before FOR.
ILLEGAL INSTRUCTION	BASIC	A statement was used which is not one of the legal BASIC statements.
ILLEGAL LINE NUMBER	BASIC	The format of the line number being used in a GOTO or IF statement is not acceptable.
ILLEGAL OPERATION	BASIC	The expression being processed does not agree with the BASIC rules (this is probably due to unmatched parentheses).
ILLEGAL SYNTAX	BASIC	The expression in a statement does not agree with the BASIC syntax.
ILLEGAL VARIABLE	BASIC	An illegal variable was used in an array.
IMPROPER ACCOUNT # ABORT ↑BS	BASIC	A user logged in under account numbers 1 (system account) or 2 (system library) and tried to run BASIC. This is prohibited.

Message	Program	Explanation
IMPROPER DIM OR RECORD STATEMENT	BASIC	Syntax error in DIM or RECORD statement, or an array name that was previously dimensioned is reused.
INVALID DEVICE NO.	BASIC	The device number in the file I/O statement is not between 0 and 11 inclusive, (or X and 11 inclusive where X is a number set by the system manager).
INVALID RECORD NO.	BASIC	The record number must be a number which is greater than or equal to 0 and less than or equal to 4095. For DECtape I/O the maximum record number is limited further by the DECtape size.
LINE TOO LONG	BASIC	Too much has been typed.
MISUSED TAB	BASIC	The TAB function was used in an invalid manner. TAB can appear only in PRINT statements.
MISUSE OF CHR$	BASIC	The CHR$ function was used in an invalid manner. CHR$, like TAB, can appear only in PRINT statements.

Message	Program	Explanation
MORE?	BASIC	Not enough values have been entered in response to an INPUT command. The rest of the values may be entered.
NEXT WITHOUT FOR	BASIC	The NEXT statement indicated has no preceding FOR statement.
NO END STATEMENT	BASIC	All programs must have an END statement.
ON INDEX OUT OF RANGE	BASIC	The value of the index is less than one, or greater than the number of statement numbers.
OUT OF DATA	BASIC	An attempt was made to READ more data than was supplied by the user.
PROGRAM IS "progname"	BASIC	This message may immediately follow an error message, to identify the current program in a series of CHAINed programs. If there is no CHAIN, this message will not occur.
PROGRAM NOT FOUND	BASIC	The file which the user tried to access with a CHAIN statement does not exist in his disk area. The PROGRAM IS message will also occur.

Message	Program	Explanation
PROGRAM TOO LARGE	BASIC	The program is too large to be executed. Make it smaller.
STACK OVERFLOW	BASIC	The user programmed a situation in which the expression is too complicated to be executed.
SUBSCRIPT ERROR	BASIC	A negative subscript was used for an array.
SYSTEM I-O ERROR	BASIC	BASIC was unable to perform the desired disk I/O.
TIME LIMIT EXCEEDED	BASIC	The number of statements executed by a job has exceeded the maximum established by the system manager. Generally, some error was made and the program is caught in a loop.
TOO MUCH INPUT, EXCESS IGNORED	BASIC	Too many values have been entered in response to an INPUT command.
UNDEFINED LINE NUMBER	BASIC	The line number appearing in a GOTO or an IF-THEN statement does not appear in the program.

Message	Program	Explanation
UNOPEN DISK UNIT	BASIC	The user tried to do a GET, PUT, or UNSAVE to device 8 or 9, without a file being previously opened on the device.
WHAT?	BASIC	The editor cannot understand the command given.
ACCOUNT ERROR	CAT	A directory listing has been requested for an account number which CAT cannot find.
ERROR IN MFD OR UFD	CAT	A directory is bad. The system may have to be rebuilt.
SYSTEM ERROR	CAT	The system has returned an error code when CAT attempted a file operation.

Message	Program	Explanation
CAN'T ASSIGN DECTAPE	COPY	An attempt has been made to use a DECtape which cannot be assigned, probably because it is assigned to someone else.
CAN'T DELETE: NAME	COPY	The named file cannot be deleted because it is protected.
DECTAPE FULL	COPY	There is not room on the DECtape for the file.
DISK FULL	COPY	The disk is full.
?ERROR	COPY	A command line may have been typed incorrectly.
SELECT ERROR	COPY	A DECtape unit is either not set to REMOTE or is WRITE PROTECTed and copy is trying to write. Type ST to try again.
COMMAND ERROR	DECODE	A command line is formatted improperly, one of the listed files cannot be found, RUBOUT has been pressed, the disk is full for output, or there has been some kind of file error.

Message	Program	Explanation
PROTECT ERROR	DECODE	An attempt has been made to write into someone else's file, write into one's own file which is protected or open to another user, or read someone else's file which is protected.
?00.00	FOCAL	Manual start given from console.
?01.00	FOCAL	Interrupt from keyboard via CTRL/C.
?01.40	FOCAL	Illegal step or line number used.
?01.78	FOCAL	Group number is too large.
?01.96	FOCAL	Double periods found in a line number.
?01.;4	FOCAL	Group zero is an illegal line number.
?01.:5	FOCAL	Line number is too large
?02.32	FOCAL	Nonexistent group referenced by DO.
?02.52	FOCAL	Nonexistent line referenced by DO.
?02.79	FOCAL	Storage was filled by push-down list.
?03.05	FOCAL	Nonexistent line used after GOTO or IF.
?03.28	FOCAL	Illegal command used.
?04.39	FOCAL	Left of = in error in FOR or SET.
?04.52	FOCAL	Excess right terminators encountered.
?04.60	FOCAL	Illegal terminator in FOR command.

Message	Program	Explanation
?04.:3	FOCAL	Missing argument in display command.
?05.48	FOCAL	Bad argument to MODIFY.
?06.06	FOCAL	Illegal use of function or number.
?06.54	FOCAL	Storage is filled by variables.
?07.22	FOCAL	Operator missing in expression or double E.
?07.38	FOCAL	No operator used before parenthesis.
?07.;6	FOCAL	Illegal function name or double operators.
?07.:9	FOCAL	No argument given after function call.
?08.47	FOCAL	Parentheses do not match.
?09.11	FOCAL	Bad argument in ERASE.
?10.:5	FOCAL	Storage was filled by text.
?11.35	FOCAL	Input buffer has overflowed.
?20.34	FOCAL	Logarithm of zero requested.
?23.36	FOCAL	Literal number is too large.
?26.99	FOCAL	Exponent is too large or negative.
?28.73	FOCAL	Division by zero requested.
?30.05	FOCAL	Imaginary square roots required.

Message	Program	Explanation
?30.71	FOCAL	Undefined library command.
?30. 0	FOCAL	Bad argument or missing argument to library command.
?31. 7	FOCAL	Illegal character, unavailable command, or unavailable function used.
?31.42	FOCAL	No such name in library directory.
?31.43	FOCAL	Attempt to enter a duplicate name in the directory.
?31.44	FOCAL	Library directory is full.
00	FORTRAN-D	Mixed mode arithmetic expression.
01	FORTRAN-D	Missing variable or constant in arithmetic expression.
03	FORTRAN-D	Comma was found in arithmetic expression.
04	FORTRAN-D	Too many operators in this expression.
05	FORTRAN-D	Function argument is in fixed-point mode.
06	FORTRAN-D	Floating-point variable used as a subscript.
07	FORTRAN-D	Too many variable names in this program.
10	FORTRAN-D	Program too large, core storage exceeded.
11	FORTRAN-D	Unbalanced right and left parentheses.

Message	Program	Explanation
12	FORTRAN-D	Illegal character found in this statement
13	FORTRAN-D	Compiler could not identify this statement
14	FORTRAN-D	More than one statement with same statement number
15	FORTRAN-D	Subscripted variable did not appear in a DIMENSION statement
16	FORTRAN-D	Statement too long to process
17	FORTRAN-D	Floating-point operand should have been fixed-point
20	FORTRAN-D	Undefined statement number
21	FORTRAN-D	Too many numbered statements in this program
22	FORTRAN-D	Too many parentheses in this statement
23	FORTRAN-D	Too many statements have been referenced before they appear in the program
25	FORTRAN-D	DEFINE statement was preceded by some executable statement
26	FORTRAN-D	Statement does not begin with a space, tab, C, or number
0240	FORTRAN-D	System file error. One of the FORTRAN components cannot be found or the disk is full, preventing FORTRAN from proceeding. Try re-calling FORT.

Message	Program	Explanation
3100	FORTRAN-D	Illegal operator on compiler stack.
3417	FORTRAN-D	Pre-precedence error.
6145	FORTRAN-D	Could not find FOSL on system device; if the error occurs, it may be necessary to reload FORT and FOSL.
6223	FORTRAN-D	Error while loading the Compiler.
6226	FORTRAN-D	Same as above.
6257	FORTRAN-D	Same as above.
6724	FORTRAN-D	No END statement on source device.
6746	FORTRAN-D	Same as above.
7114	FORTRAN-D	Same as above.
01		Checksum error on FORTRAN binary input
02		Illegal origin or data address on FORTRAN binary input
04		Disk input-output error
05		High-speed reader error
06		Illegal FORTRAN binary input device
11		Attempt to divide by zero
12		Floating-point input data conversion error.
13		Illegal op code
14		Disk input-output error

Message	Program	Explanation
15		Non-FORMAT statement used as a FORMAT
16		Illegal FORMAT specification
17		Floating-point number larger than 2047
20		Square root of a negative number
21		Exponential negative number
22		Logarithm of a number larger than 2047
40		Illegal device code used in READ or WRITE statement
41		System device full, could not complete a WRITE statement
76		Stack underflow error
77		Stack overflow error
LOAD ERROR	LOGID	The file given as input is not a valid binary file.
DUPLICATE ACCOUNT #	LOGID	An attempt has been made to define an account, but the account number already exists with a different password.
LOGGED IN -- NOT DELETED	LOGID	A user is logged in under an account which the manager is attempting to delete.
NOT FOUND	LOGID	An attempt has been made to delete an account which cannot be found.

Message	Program	Explanation
OPEN FILE - NOT DELETED	LOGID	An account cannot be deleted because one of its files is open to some other user.
RESTRICTED ACCESS	LOGID	Accounts 1, 2, and 3 cannot be deleted by LOGID.
RESTRICTED ACCESS ↑BS	LOGID	LOGID can be run only under account 1.
SYSTEM ERROR	LOGID	An error code has been returned from a file operation. Possibly someone has a file open belonging to an account which is being deleted.

Message	Program	Explanation
ALREADY LOGGED IN	MONITOR	The user tried to log in on a console which is already in use.
BAD DIRECTORY	MONITOR	A request cannot be honored because a disk directory is invalid.
BAD FILE NAME	MONITOR	An invalid file name has been given
BUSY	MONITOR	The user attempted to talk to a console which is currently printing or on which another user is typing.
DEVICE NOT AVAILABLE	MONITOR	A device which a user tried to assign is not present on the system, or is temporarily busy and will be free in a few seconds.

Message	Program	Explanation
DIRECTORY FULL	MONITOR	A request cannot be honored because the user's directory is full, or his disk quota is exceeded.
DISK FULL	MONITOR	A request cannot be honored because the disk is full.
DISK ERROR FOR JOB	MONITOR	There has been an error while reading or writing in a disk file.
MYFILE EXCEEDING DISK QUOTA	MONITOR	The user has extended a file beyond the allowed disk quota. The amount of extra space (grace) the user is allowed is determined by the system manager. This message is informational only.
FAILED BY n SEGMENTS	MONITOR	An EXTEND command cannot be completed because the disk or directory is full, or the disk quota would be exceeded.
FILE IN USE	MONITOR	A file cannot be altered because it is open to another user, or possibly twice to one user.
FILE NOT FOUND	MONITOR	A file cannot be found.
FILE NOT OPEN	MONITOR	A file request has been made, but there is no file open on the internal file number which has been given.
FULL	MONITOR	The system is full. Another user cannot log in until one of the present users logs out.
HUNG DEVICE FOR JOB	MONITOR	A device which the user tried to use is not responding. For the paper tape punch and line printer, this message will be repeated until the device is turned on or released.

Message	Program	Explanation
ILLEGAL IOT FOR JOB	MONITOR	The user has tried to execute an IOT which is illegal. This can mean that either he has tried to use a device which is not available, or he has executed a privileged IOT, but is not in the privileged condition at the time.
ILLEGAL REQUEST	MONITOR	The user requested an illegal command. This error usually results when some parameter has been given an incorrect value or the request refers to a facility not owned by the user.
LOGIN PLEASE	MONITOR	The user attempted to use a console which is not logged into the system.
PROTECTION VIOLATION	MONITOR	A request cannot be honored because of a file's protection, or because it is open more than once.
SWAP ERROR FOR JOB	MONITOR	These has been a disk error while swapping the user's program.

Message	Program	Explanation
TYPE ↑BS FIRST	MONITOR	The user typed a command which cannot be honored, while a program is running. The user should type CTRL/B followed by S, a carriage return, and then enter his command. He may then type START to continue running the program.
UNAUTHORIZED ACCOUNT	MONITOR	The user attempted to log into the system with an invalid account number or password.
WAIT FOR I/O	MONITOR	A command cannot be honored because of I/O in progress. Wait a few seconds and try again.
LC	Non-Fatal BASIC	An invalid character was typed in response to an INPUT statement.
LN	Non-Fatal	An attempt was made to compute the logarithm of zero or a negative number. Zero is used for the result.
OV	Non-Fatal BASIC	Overflow - the result of a calculation was too large for the computer to handle. The largest possible number is used for the result.

Message	Program	Explanation
PW	Non-Fatal BASIC	An attempt was made to raise a negative number to a fractional power. The absolute value of that number raised to the fractional power is used.
SQ	Non-Fatal BASIC	An attempt was made to compute the square root of a negative number. The square root of the absolute value is used for the result.
UN	Non-Fatal BASIC	Underflow - the result of a calculation was too small for the computer to handle. Zero is used for the result.
/0	Non-Fatal BASIC	Zero divide - an attempt was made to divide by zero. The largest possible number is used for the result.
BE	PAL-D	Two PAL-D internal tables have overlapped. This situation can usually be corrected by decreasing the level of literal nesting or number of current page literals used prior to this point on the page.

Message	Program	Explanation
DE	PAL-D	System device error - An error was detected when trying to read or write onto the system device; after three failures, control is returned to the Monitor.
DF	PAL-D	Systems device full - The capacity of the system device has been exceeded; assembly is terminated and control is returned to the Monitor.
IC	PAL-D	Illegal character - An illegal character was encountered other than in a comment or TEXT field; the character is ignored and the assembly continued.
ID	PAL-D	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	PAL-D	Illegal equals - An equal sign was used in the wrong context. Examples: TAD A+=B — the expression to the left of the equal sign is not a single symbol A+B=C — or, the expression to the right of the equal sign was not previously defined.

Message	Program	Explanation
II	PAL-D	Illegal indirect - An off-page reference was made; a link could not be generated because the indirect bit was already set.
ND	PAL-D	The program terminator, $, is missing.
PE	PAL-D	Current nonzero page exceeded - An attempt was made to: a. override a literal with an instruction, or b. override an instruction with a literal; this can be corrected by (1) decreasing the number of literals on the page or (2) decreasing the number of instructions on the page.
SE	PAL-D	Symbol table exceeded - Assembly is terminated and control is returned to the Monitor.
US	PAL-D	Undefined symbol - A symbol has been processed during pass 2 that was not defined before the end of pass 1.

Message	Program	Explanation
ZE	PAL-D	Page 0 exceeded - Same as PE except with reference to page 0.
COMMAND ERROR	PIP	An illegal option has been entered.
DISK I/O ERROR	PIP	Self-explanatory.
DIRECTORY FULL	PIP	The file specified for output cannot be written into because the user's directory is full.
DEVICE NOT AVAILABLE OR HUNG	PIP	A device cannot be assigned or is hung. Hung devices usually result from having the device turned off.
FILE NOT FOUND	PIP	The file listed as input cannot be found.
LOAD ERROR (nnnn)	PIP	A SAVE format paper tape was not read properly. nnnn = the final checksum.
OUTPUT FILE IN USE	PIP	The output file cannot be written into because some other user has that file open.

Message	Program	Explanation
PROTECTED	PIP	A file cannot be accessed because of protection.
SYSTEM ERROR DISK FULL	PIP	The disk is full.
ARE YOU SURE?	PUTR	The user is trying to zero a device. To proceed, type Y. Type anything else otherwise.
BAD CARD TRY AGAIN?	PUTR	When PUTR tried to read a card, it did not receive 39, 40, or 80 columns. Typing N and a carriage return will cause PUTR to close the output file, or typing anything else will cause PUTR to try reading once again.
BAD INPUT DEVICE	PUTR	An attempt has been made to input from a device which is output only, such as LPT:.
BAD OUTPUT DEVICE	PUTR	An attempt has been made to output to an input-only device, such as PTR:.
BAD SWITCH	PUTR	A switch (characters following a slash) is invalid.
BAD SYSTEM DIRECTORY	PUTR	The system directory is invalid.
CAN'T ASSIGN DEVICE	PUTR	The listed device cannot be assigned. Type any character and try again, or type CTRL/C to give up.

Message	Program	Explanation
CAN'T DELETE	PUTR	The listed file could not be deleted.
CAN'T OPEN INPUT	PUTR	The file listed as input cannot be opened.
CAN'T OPEN OUTPUT	PUTR	The desired output file cannot be opened.
CREATE ERROR	PUTR	A file with the desired name cannot be created on the system disk. Perhaps one with that name already exists and is protected.
DECTAPE STATUS B ERROR	PUTR	An error has occurred while reading or writing a DECtape. The error code is in the switch register, which can be found by typing CTRL/B, then W, then RETURN
DIRECTORY FULL	PUTR	The DECtape or RKØ5 directory has no more room.
ILLEGAL SYNTAX	PUTR	PUTR cannot understand the command which was typed.
ILLEGAL UNIT	PUTR	The unit number for DECtape or RK8E was invalid.
INPUT ERROR	PUTR	There has been an error while reading from the input device.
LINE TOO LONG	PUTR	The command line just typed was too long for PUTR.

Message	Program	Explanation
NOT UNDER ACCOUNT 1	PUTR	PUTR must never be run under account 1.
NO END OF FILE	PUTR	When reading a BASIC file, the physical end of the file was found before the logical end of file.
NO FILES FOUND	PUTR	The user has requested some operation, but PUTR has found no files to operate upon.
OLD DECTAPE	PUTR	An attempt has been made to output to a DECtape which is neither OS/8 nor PUTR format.
OUTPUT FILE TOO LARGE	PUTR	There is not room for the output file.
PUTR2 APPEND ERROR	PUTR2	PUTR2 could not successfully append itself to PUTR. Perhaps PUTR is being used by another user, has already had PUTR2 appended to it, is not the same version as PUTR2, or the disk is full.
RKØ5 I/O ERROR	PUTR	There has been an error while reading or writing on the RKØ5. The status can be found in the switch register, which can be found by typing CTRL/B followed by SW and RETURN.
RKØ5 NOT READY?	PUTR	The RKØ5 is not ready or write-locked. Type CTRL/C to abort the operation or any other character to try again.
SELECT ERROR?	PUTR	A DECtape unit is not on remote or not write enabled. Type a carriage return to try again or CTRL/C to abort the operation.
SYSTEM READ ERROR	PUTR	There has been an error while reading from the system disk.
SYSTEM WRITE ERROR	PUTR	There has been an error while writing to the system disk.
USE DEL *	PUTR	Zero is an invalid command for SYS:.
WHAT?	PUTR	PUTR cannot understand the command just typed.

HOW TO OBTAIN SOFTWARE INFORMATION

SOFTWARE NEWSLETTERS, MAILING LIST

The Software Communications Group, located at corporate headquarters in Maynard, publishes software newsletters for the various DIGITAL products. Newsletters are published monthly, and keep the user informed about customer software problems and solutions, new software products, documentation corrections, as well as programming notes and techniques.

There are two similar levels of service:

- The Software Dispatch
- The Digital Software News

The Software Dispatch is part of the Software Maintenance Service. This service applies to the following software products:

PDP-9/15
RSX-11D
DOS/BATCH
RSTS-E
DECsystem-10

A Digital Software News for the PDP-11 and a Digital Software News for the PDP-8/12 are available to any customer who has purchased PDP-11 or PDP-8/12 software.

A collection of existing problems and solutions for a given software system is published periodically. A customer receives this publication with his initial software kit with the delivery of his system. This collection would be either a Software Dispatch Review or Software Performance Summary depending on the system ordered.

A mailing list of users who receive software newsletters is also maintained by Software Communications. Users must sign-up for the newsletter they desire. This can be done by either completing the form supplied with the Review or Summary or by writing to:

Software Communications
P.O. Box F
Maynard, Massachusetts 01754

SOFTWARE PROBLEMS

Questions or problems relating to DIGITAL's software should be reported as follows:

North and South American Submitters:

Upon completion of Software Performance Report (SPR) form remove last copy and send remainder to:

Software Communications
P.O. Box F
Maynard, Massachusetts 01754

The acknowledgement copy will be returned along with a blank SPR form upon receipt. The acknowledgement will contain a DIGITAL assigned SPR number. The SPR number or the preprinted number should be referenced in any future correspondence. Additional SPR forms may be obtained from the above address.

All International Submitters:

Upon completion of the SPR form, reserve the last copy and send the remainder to the SPR Center in the nearest DIGITAL office. SPR forms are also available from our SPR Centers.

PROGRAMS AND MANUALS

Software and manuals should be ordered by title and order number. In the United States, send orders to the nearest distribution center.

Digital Equipment Corporation
Software Distribution Center
146 Main Street
Maynard, Massachusetts 01754

Digital Equipment Corporation
Software Distribution Center
1400 Terra Bella
Mountain View, California 94043

Outside of the United States, orders should be directed to the nearest Digital Field Sales Office or representative.

USERS SOCIETY

DECUS, Digital Equipment Computers Users Society, maintains a user exchange center for user-written programs and technical application information. The Library contains approximately 1,900 programs for all DIGITAL computer lines. Executive routines, editors, debuggers, special functions, games, maintenance and various other classes of programs are available.

DECUS Program Library Catalogs are routinely updated and contain lists and abstracts of all programs according to computer line:

- PDP-8, FOCAL-8, BASIC-8, PDP-12
- PDP-7/9, 9, 15
- PDP-11, RSTS-11
- PDP-6/10, 10

Forms and information on acquiring and submitting programs to the DECUS Library may be obtained from the DECUS office.

In addition to the catalogs, DECUS also publishes the following:

DECUSCOPE	-The Society's technical newsletter, published bi-monthly, aimed at facilitating the interchange of technical information among users of DIGITAL computers and at disseminating news items concerning the Society. Circulation reached 19,000 in May, 1974.
PROCEEDINGS OF THE DIGITAL EQUIPMENT USERS SOCIETY	-Contains technical papers presented at DECUS Symposia held twice a year in the United States, once a year in Europe, Australia, and Canada.
MINUTES OF THE DECsystem-10 SESSIONS	-A report of the DECsystem-10 sessions held at the two United States DECUS Symposia.
COPY-N-Mail	-A monthly mailed communique among DECsystem-10 users.
LUG/SIG	-Mailing of Local User Group (LUG) and Special Interest Group (SIG) communique, aimed at providing closer communication among users of a specific product or application.

Further information on the DECUS Library, publications, and other DECUS activities is available from the DECUS offices listed below:

DECUS
Digital Equipment Corporation
146 Main Street
Maynard, Massachusetts 01754

DECUS EUROPE
Digital Equipment Corp. International
(Europe)
P.O. Box 340
1211 Geneva 26
Switzerland

Please cut along this line.

READER'S COMMENTS

NOTE: This form is for document comments only. Problems with software should be reported on a Software Problem Report (SPR) form (see the HOW TO OBTAIN SOFTWARE INFORMATION page).

Did you find errors in this manual? If so, specify by page.

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

Is there sufficient documentation on associated system programs required for use of the software described in this manual? If not, what material is missing and where should it be placed?

Please indicate the type of user/reader that you most nearly represent.

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Non-programmer interested in computer concepts and capabilities

Name _____ Date _____

Organization _____

Street _____

City _____ State _____ Zip Code or Country _____

If you do not require a written reply, please check here. ☐

---------- Fold Here ----------

---------- Do Not Tear - Fold Here and Staple ----------

FIRST CLASS
PERMIT NO. 33
MAYNARD, MASS.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

Postage will be paid by:

digital

Software Communications
P. O. Box F
Maynard, Massachusetts 01754

digital equipment corporation