digital

software

OS/78
User's Manual

Order No. DEC-S8-OS78A-A-D

PDP8

more than 30,000 installed worldwide

OS/78
User's Manual

Order No. DEC-S8-OS78A-A-D

September 1977

This document is the user's manual for the OS/78 V1 operating system. Its purpose is to acquaint new users with the operating system designed for the DECstation 78 minicomputer. This manual presents the background material for getting on the air, and a detailed description of the OS/78 commands that are used to direct computer operations. It also describes the OS/78 Editor, PAL8 assembly language, and the two high-level languages supported by the system, BASIC and FORTRAN IV. Command examples and demonstration programs are contained throughout the manual.

SUPERSESSION/UPDATE INFORMATION: This is a new manual.

OPERATING SYSTEM AND VERSION: OS/78 V1

To order additional copies of this document, contact the Software Distribution Center, Digital Equipment Corporation, Maynard, Massachusetts 01754.

digital equipment corporation · maynard, massachusetts

First Printing, September 1977

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 be used or copied only in accordance with 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.

Copyright © 1977 by Digital Equipment Corporation

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
DEC
PDP
DECUS
UNIBUS
COMPUTER LABS
COMTEX
DDT
DECCOMM
DECsystem-10
DECtape
DIBOL
EDUSYSTEM
FLIP CHIP
FOCAL
INDAC
LAB-8
DECSYSTEM-20
MASSBUS
OMNIBUS
OS/8
OS/78
PHA
RSTS
RSX
TYPESET-8
TYPESET-10
TYPESET-11

CONTENTS

			Page
PREFACE			xv
CHAPTER 1		**INTRODUCTION**	**1-1**
	1.1	DECSTATION 78 MINICOMPUTER SYSTEM	1-1
	1.2	OS/78 SOFTWARE	1-1
CHAPTER 2		**GETTING ON THE AIR**	**2-1**
	2.1	STARTING THE SYSTEM	2-1
	2.1.1	Loading the Diskettes	2-1
	2.1.2	Calling the Keyboard Monitor	2-1
	2.1.3	Using the Keyboard	2-1
	2.1.4	Typing Convention and Symbology	2-2
	2.2	MAKING A BACKUP COPY	2-3
	2.3	DEMONSTRATION PROGRAMS	2-3
	2.4	NAMING DEVICES AND FILES	2-3
	2.4.1	Devices	2-3
	2.4.2	File Names and Extensions	2-4
	2.5	ENTERING MONITOR COMMANDS	2-4
	2.5.1	OS/78 Command Format	2-5
	2.5.2	Incorrect Commands	2-6
	2.5.3	Correcting Errors	2-7
	2.5.4	Using Input/Output Options	2-7
	2.5.5	Remembering Previous Arguments	2-8
	2.5.6	Using Wildcards	2-9
	2.5.6.1	Wildcard Input File Specifications	2-10
	2.5.6.2	Wildcard Output File Specifications	2-10
	2.5.7	Indirect Commands (@ Construction)	2-11
	2.5.8	Asterisk (*)	2-11
	2.6	USING DEFAULTS	2-12
	2.7	WHERE TO GET MORE INFORMATION	2-12
	2.8	FILES AND FILE-STRUCTURED DEVICES	2-13
	2.8.1	File-Structured Devices	2-13
	2.8.2	File Directories	2-14
	2.8.3	File Types	2-14
	2.8.4	File Formats	2-14
	2.9	OS/78 COMMAND SUMMARY	2-15
CHAPTER 3		**OS/78 COMMANDS**	**3-1**
	3.1	ASSIGN COMMAND	3-2
	3.2	BASIC COMMAND	3-3
	3.3	COMPARE COMMAND	3-4
	3.3.1	Comparing Part of a Line	3-5
	3.3.2	Making Changes While Using the COMPARE Command	3-5
	3.4	COMPILE COMMAND	3-7
	3.5	COPY COMMAND	3-8
	3.5.1	File Transfer	3-8
	3.5.2	Examples of Copy Commands	3-8
	3.5.3	Predeletion	3-9

CONTENTS (CONT.)

		Page
3.5.4	Postdeletion	3-9
3.5.5	Considering the Date on a Transfer	3-9
3.5.6	File Protection During a Transfer Operation	3-10
3.5.7	Terminating Copy Operations	3-11
3.6	CREATE COMMAND	3-14
3.7	CREF COMMAND	3-15
3.8	DATE COMMAND	3-17
3.9	DEASSIGN COMMAND	3-18
3.10	DELETE COMMAND	3-19
3.11	DIRECT COMMAND	3-21
3.11.1	Considering the Date in a Directory Listing	3-22
3.11.2	Choosing the Directory Format	3-22
3.12	DUPLICATE COMMAND	3-24
3.12.1	Changing Devices Before and After Executing the DUPLICATE Command	3-24
3.12.2	Performing a Read Check	3-24
3.12.3	Transfer Without Checking for Identical Contents	3-24
3.12.4	Check for Identical Contents Without Transferring	3-25
3.13	EDIT COMMAND	3-26
3.14	EXECUTE COMMAND	3-28
3.15	GET COMMAND	3-29
3.16	HELP COMMAND	3-30
3.17	LIST COMMAND	3-32
3.18	LOAD COMMAND	3-34
3.18.1	Absolute Binary Files	3-34
3.18.2	Loading Programs in Specified Areas	3-34
3.18.3	Editing or Patching a SAVE File	3-35
3.18.4	Executive After Loading	3-35
3.18.5	Clearing Memory After Loading	3-35
3.18.6	Relocatable FORTRAN Binary Files	3-35
3.19	MAP COMMAND	3-36
3.20	MEMORY COMMAND	3-39
3.21	ODT COMMAND	3-40
3.22	PAL COMMAND	3-41
3.23	R COMMAND	3-42
3.24	RENAME COMMAND	3-43
3.25	RUN COMMAND	3-44
3.26	SAVE COMMAND	3-45
3.27	SET COMMAND	3-48
3.27.1	ECHO	3-48
3.27.2	ARROW	3-48
3.27.3	SCOPE	3-49
3.27.4	WIDTH = n	3-49
3.27.5	HEIGHT m	3-50
3.27.6	PAGE	3-50
3.27.7	ESC	3-50
3.27.8	PAUSE	3-50
3.27.9	READ ONLY	3-50
3.27.10	COL n	3-51

CONTENTS (CONT.)

		Page
3.27.11	LC	3-51
3.27.12	INIT xxxxx	3-51
3.28	START COMMAND	3-53
3.29	SQUISH COMMAND	3-54
3.30	SUBMIT COMMAND	3-55
3.30.1	Processing and Terminating a Batch Input File	3-55
3.30.2	Spooling	3-56
3.31	TERMINATE COMMAND	3-57
3.32	TYPE COMMAND	3-58
3.33	UA, UB, UC COMMANDS	3-60
3.34	ZERO COMMANDS	3-61
CHAPTER 4	**CREATING AND EDITING YOUR PROGRAM**	**4-1**
4.1	INTRODUCTION	4-1
4.2	PAGE MAKEUP	4-1
4.3	TEXT BUFFER	4-1
4.4	CALLING THE EDITOR	4-2
4.5	MODES OF OPERATION	4-2
4.6	EDITOR KEY COMMANDS	4-2
4.7	SPECIAL CHARACTERS	4-3
4.7.1	DELETE Key	4-3
4.7.2	Period (.)	4-3
4.7.3	Slash (/)	4-4
4.7.4	LINE FEED Key	4-4
4.7.5	Right-Angle Bracket (>)	4-4
4.7.6	Left-Angle Bracket (<)	4-4
4.7.7	Equal Sign (=)	4-4
4.7.8	Colon (:)	4-5
4.7.9	Tabulation (TAB)	4-5
4.7.10	ESCape Key	4-5
4.7.11	Lower Case Characters	4-5
4.8	EDITOR COMMANDS	4-5
4.8.1	Input Commands	4-5
4.8.2	List Commands	4-6
4.8.3	Output Commands	4-7
4.8.4	Editing Commands	4-9
4.8.5	Search Commands	4-9
4.9	CREATING A NEW FILE	4-10
4.10	PAGING FILES	4-10
4.11	SEARCHING THROUGH FILES	4-11
4.11.1	Single Character Search	4-11
4.11.1.1	nS Command	4-11
4.11.1.2	m,nS Command	4-11
4.11.1.3	S Command	4-11
4.11.2	Character String Search	4-12
4.11.2.1	IntraBuffer Character String Search	4-12
4.11.2.2	InterBuffer Character String Search	4-14
4.12	EDITING AN EXAMPLE PROGRAM	4-15
4.13	EDITOR ERROR MESSAGES	4-18
4.14	SUMMARY OF EDITOR COMMANDS AND SPECIAL CHARACTERS	4-19

CONTENTS (CONT.)

			Page
CHAPTER 5		**THE PAL8 ASSEMBLER**	**5-1**
	5.1	INTRODUCTION	5-1
	5.2	CREATING AND RUNNING AN ASSEMBLY PROGRAM	5-4
	5.2.1	Creating a Program	5-4
	5.2.2	Assembling a Program	5-5
	5.2.3	Loading and Saving a Program	5-6
	5.2.4	Executing the Program	5-6
	5.2.5	Getting and Using a Cross-Reference Listing	5-7
	5.2.6	Obtaining a Memory Map	5-7
	5.3	PAL8 OPTIONS	5-8
	5.4	CHARACTER SET	5-9
	5.5	STATEMENTS	5-10
	5.5.1	Labels	5-10
	5.5.2	Instructions	5-10
	5.5.3	Operands	5-10
	5.5.4	Comments	5-10
	5.6	FORMAT CHARACTERS	5-10
	5.6.1	Form Feed	5-10
	5.6.2	Tab	5-11
	5.6.3	Statement Terminators	5-11
	5.7	NUMBERS	5-11
	5.8	SYMBOLS	5-12
	5.8.1	Permanent Symbols	5-12
	5.8.2	User-Defined Symbols	5-12
	5.8.3	Current Location Counter	5-12
	5.8.4	Symbol Table	5-13
	5.8.5	Direct Assignment Statements	5-14
	5.8.6	Symbolic Instructions	5-15
	5.8.7	Symbolic Operands	5-15
	5.9	EXPRESSIONS	5-15
	5.9.1	Operators	5-15
	5.9.2	Special Characters	5-18
	5.9.2.1	Dot (.)	5-19
	5.9.2.2	Double Quote (")	5-19
	5.9.2.3	Parentheses ( ) and Brackets [ ]	5-19
	5.9.2.4	Angle Brackets (<>)	5-20
	5.9.2.5	Dollar Sign ($)	5-21
	5.10	INSTRUCTION SET	5-21
	5.10.1	Memory Reference Instructions	5-21
	5.10.2	Microinstructions	5-22
	5.10.3	Autoindexing	5-25
	5.11	PSEUDO-OPERATORS	5-25
	5.11.1	Indirect and Page Zero Addressing	5-25
	5.11.2	Extended Memory	5-26
	5.11.3	Resetting the Location Counter	5-26
	5.11.4	Reserving Memory	5-27
	5.11.5	Relocation Pseudo-Operator	5-27
	5.11.6	Suppressing the Listing	5-27
	5.11.7	Controlling Page Format	5-28

CONTENTS (CONT.)

Page

5.11.8 Altering the Permanent Symbol Table . . . 5-28
5.11.9 Conditional Assembly Pseudo-Operators . . . 5-29
5.11.10 Radix Control . . . 5-29
5.11.11 Entering Text Strings . . . 5-30
5.11.12 End-of-File Signal . . . 5-30
5.11.13 Use of DEVICE and FILENAME Pseudo-Operators . . . 5-30
5.12 LINK GENERATION AND STORAGE . . . 5-31
5.13 TERMINATING ASSEMBLY . . . 5-32
5.14 ERROR MESSAGES . . . 5-32
5.15 PAL8 PERMANENT SYMBOL TABLE . . . 5-34

CHAPTER 6 BASIC . . . 6-1
6.1 INTRODUCTION . . . 6-1
6.2 MAJOR COMPONENTS OF OS/78 BASIC . . . 6-1
6.3 BASIC INSTRUCTION REPERTOIRE . . . 6-1
6.4 CALLING BASIC . . . 6-2
6.5 BASIC EDITOR COMMANDS . . . 6-2
6.5.1 Using the BASIC Editor . . . 6-2
6.5.2 BASIC Editor Commands . . . 6-2
6.5.2.1 NEW Command . . . 6-2
6.5.2.2 OLD Command . . . 6-4
6.5.2.3 LIST/LISTNH Commands . . . 6-4
6.5.2.4 SAVE Command . . . 6-5
6.5.2.5 RUN/RUNNH Commands . . . 6-6
6.5.2.6 NAME Command . . . 6-6
6.5.2.7 SCRATCH Command . . . 6-6
6.5.2.8 BYE Command . . . 6-6
6.5.3 BASIC Control Keys . . . 6-7
6.5.3.1 Correcting Typing and Format Errors (DELETE, CTRL/U) . . . 6-7
6.5.3.2 Eliminating and Substituting Program Lines (RETURN) . . . 6-7
6.5.3.3 Interrupting Program Execution . . . 6-7
6.5.3.4 Controlling Program Listings at the Terminal Console (CTRL/S, CTRL/C and CTRL/O) . . . 6-7
6.5.4 Resequencing Programs (RESEQ) . . . 6-7
6.6 DATA FORMATS ACCEPTABLE TO BASIC . . . 6-8
6.6.1 Numeric Information . . . 6-8
6.6.2 Numbers . . . 6-8
6.6.3 Simple Variables . . . 6-9
6.6.4 Subscripted Variables . . . 6-10
6.6.5 Arithmetic Operations . . . 6-10
6.6.5.1 Operators . . . 6-10
6.6.5.2 Priority . . . 6-10
6.6.5.3 Parentheses . . . 6-11
6.6.5.4 Rules for Exponentiation . . . 6-11
6.6.5.5 Relational Operators . . . 6-12
6.6.6 String Information . . . 6-12
6.6.6.1 String Character Set . . . 6-12
6.6.6.2 String Conventions . . . 6-13
6.6.6.3 String Concatenation . . . 6-14
6.6.7 Format Control Characters . . . 6-14
6.6.8 Files . . . 6-15

CONTENTS (CONT.)

Page

6.7 BASIC STATEMENTS 6-15
6.7.1 Statement Line Numbers (Sequencing) 6-15
6.7.2 The PRINT Statement 6-16
6.7.3 Information Entry Statements (DATA, READ) 6-18
6.7.3.1 DATA Statement Format 6-18
6.7.3.2 READ Statement Format 6-18
6.7.4 LET Statement 6-20
6.7.5 Loops (FOR and NEXT Statement) 6-21
6.7.5.1 FOR Statement Format 6-21
6.7.5.2 NEXT Statement Format 6-24
6.7.6 Control Statements (GOTO, IF-THEN, GOSUB, RETURN) 6-24
6.7.6.1 Unconditional Branch (GOTO Statement) 6-24
6.7.6.2 Conditional Branch (IF-THEN Statement) 6-25
6.7.6.3 Branch to Subroutine (GOSUB Statements) 6-26
6.7.6.4 Return from Subroutine (RETURN Statement) 6-28
6.7.7 Program Termination Statements (END, STOP) 6-28
6.7.7.1 END Statement Format 6-28
6.7.7.2 STOP Statement Format 6-28
6.7.8 The INPUT Statement 6-28
6.7.9 The REMark Statement 6-30
6.7.10 Ancillary Statements (DIMension, RESTORE, DEFine, RANDOMIZE and CHAIN) 6-31
6.7.10.1 DIMension Statement Format 6-31
6.7.10.2 RESTORE Statement 6-33
6.7.10.3 DEFine Statement 6-34
6.7.10.4 RANDOMIZE Statement 6-35
6.7.10.5 CHAIN Statement 6-36
6.7.11 File Handling Statements 6-36
6.7.11.1 FILE# Statement 6-36
6.7.11.2 PRINT# Statement 6-37
6.7.11.3 INPUT# Statement 6-39
6.7.11.4 RESTORE# Statement 6-40
6.7.11.5 CLOSE# Statement 6-41
6.7.11.6 IF END# Statement 6-42
6.8 BASIC FUNCTIONS 6-42
6.8.1 Arithmetic Functions (ABS, INT, EXP, RND, SGN, SQR) 6-43
6.8.1.1 BASIC ABS Function 6-43
6.8.1.2 BASIC EXP Function 6-43
6.8.1.3 BASIC INT Function 6-43
6.8.1.4 BASIC RND Function 6-43
6.8.1.5 BASIC SGN Function 6-44
6.8.1.6 BASIC SQR Function 6-44
6.8.2 Trigonometric Functions (ATN, COS, LOG, SIN) 6-45
6.8.2.1 BASIC ATN Function 6-45
6.8.2.2 BASIC COS Function 6-45
6.8.2.3 BASIC LOG Function 6-45
6.8.2.4 BASIC SIN Function 6-45
6.8.3 String Handling Functions (ASC, CHP$, DAT$, LEN, POS, SEG$, STR$, TAB, VAL) 6-45
6.8.3.1 BASIC ASC Function 6-45

CONTENTS (CONT.)

Page

6.8.3.2 BASIC CHR$ Function 6-47
6.8.3.3 BASIC DAT$ Function 6-47
6.8.3.4 BASIC LEN Function 6-48
6.8.3.5 BASIC POS Function 6-48
6.8.3.6 BASIC SEG$ Function 6-48
6.8.3.7 BASIC STR$ Function 6-49
6.8.3.8 BASIC TAB Function 6-49
6.8.3.9 BASIC VAL Function 6-50
6.8.4 Display Console Control Function (PNT) 6-51
6.8.5 Trace Function 6-52
6.9 SUMMARY OF BASIC EDITOR COMMANDS 6-53
6.10 SUMMARY OF BASIC STATEMENTS 6-53
6.11 SUMMARY OF BASIC FUNCTIONS 6-55
6.12 BASIC ERROR MESSAGES 6-57
6.12.1 Compiler Error Messages 6-57
6.12.2 Run-Time System Error Messages 6-57

CHAPTER 7 OS/78 FORTRAN IV 7-1
7.1 SYSTEM OVERVIEW 7-1
7.1.1 Source Programs 7-1
7.1.2 Using the EXECUTE Command 7-1
7.1.3 Compiling 7-2
7.1.4 Loading 7-2
7.1.5 Run-Time System 7-4
7.1.6 System Library 7-4
7.2 OS/78 FORTRAN IV OPERATION 7-4
7.2.1 FORTRAN IV Compiler 7-5
7.2.1.1 Using the COMPILE Command 7-5
7.2.1.2 Examples of Compilation 7-6
7.2.1.3 Compiler Error Messages 7-6
7.2.2 FORTRAN IV Loader 7-8
7.2.2.1 Examples of Loading 7-9
7.2.2.2 Loader Error Messages 7-9
7.2.3 FORTRAN IV Run-Time System (FRTS) 7-10
7.2.3.1 Run-Time System Error Messages 7-15
7.2.4 FORTRAN IV Library: Library Functions and Subroutines 7-16
7.3 RUNNING OS/78 FORTRAN IV PROGRAMS 7-22
7.3.1 Executing a Program 7-22
7.3.2 Compiling a Program 7-23
7.3.3 Creating a Loader Image File 7-24
7.3.4 Running Subprograms 7-24
7.3.5 Specifying I/O Devices 7-27
7.4 FORTRAN IV SOURCE LANGUAGE 7-27
7.4.1 Constants 7-28
7.4.1.1 Integer Constants 7-28
7.4.1.2 Real Constants 7-28
7.4.1.3 Octal Constants 7-29
7.4.1.4 Logical Constants 7-29
7.4.1.5 Hollerith Constants 7-29
7.4.2 Variables 7-29

CONTENTS (CONT.)

			Page
	7.4.2.1	Arrays	7-30
	7.4.2.2	Subscripts	7-30
	7.4.3	Expressions	7-30
	7.4.3.1	Arithmetic Expressions	7-30
	7.4.3.2	Logical Expressions	7-32
	7.4.4	Assignment Statements	7-34
	7.4.4.1	Arithmetic Statements	7-34
	7.4.4.2	The GO TO Assignment Statement	7-36
	7.4.5	Control Statements	7-36
	7.4.5.1	GO TO Statements	7-36
	7.4.5.2	IF Statements	7-38
	7.4.5.3	DO Statement	7-39
	7.4.5.4	CONTINUE Statement	7-41
	7.4.5.5	PAUSE Statement	7-42
	7.4.5.6	STOP Statement	7-42
	7.4.5.7	END Statement	7-42
	7.4.6	Data Transmission Statements	7-42
	7.4.6.1	FORMAT Statement	7-42
	7.4.6.2	DEFINE FILE Statement	7-50
	7.4.6.3	Input/Output Statements	7-50
	7.4.6.4	Device Control Statements	7-54
	7.4.7	Specification Statements	7-54
	7.4.7.1	Storage Specification Statements Dimension Statement	7-54
	7.4.7.2	Type Declaration Statements	7-59
	7.4.8	Subprogram Statements	7-59
	7.4.8.1	Functions	7-59
	7.4.8.2	FUNCTION Statements	7-60
	7.4.8.3	Subroutine Subprograms	7-61
	7.4.8.4	RETURN Statement	7-62
	7.4.8.5	BLOCK DATA Statement	7-62
	7.4.8.6	EXTERNAL Statement	7-63
	7.4.9	OS/78 FORTRAN IV Statement Summary	7-63
CHAPTER	**8**	**BATCH**	**8-1**
	8.1	BATCH PROCESSING UNDER OS/78	8-1
	8.2	BATCH MONITOR COMMANDS	8-2
	8.3	THE BATCH INPUT FILE	8-4
	8.4	BATCH ERROR MESSAGES	8-6
	8.5	RESTRICTIONS UNDER OS/78 BATCH	8-7
CHAPTER	**9**	**DEBUGGING A PROGRAM**	**9-1**
	9.1	ODT FEATURES	9-1
	9.2	CALLING AND USING ODT	9-1
	9.3	ODT COMMANDS	9-2
	9.3.1	Special Characters	9-2
	9.3.1.1	Slash(/) – Open This Location	9-2
	9.3.1.2	RETURN – Close Location	9-3
	9.3.1.3	LINE FEED – Close Location, Open Next Location	9-3
	9.3.1.4	Circumflex (^) – Close Location, Take Contents as Memory Reference Instruction and Open Effective Address	9-4
	9.3.1.5	Underline (_) – Close Location, Open Indirectly	9-4
	9.3.2	Illegal Characters	9-4

CONTENTS (CONT.)

		Page
9.3.3	Control Commands	9-4
9.3.3.1	fnnnnG – Transfer Control to Program at Location nnnn of field f	9-4
9.3.3.2	fnnnnB – Set Breakpoint at Location nnnn of field f	9-4
9.3.3.3	B – Remove Breakpoint	9-5
9.3.3.4	A – Open C(AC) Location	9-4
9.3.3.5	L – Open C(L) Location	9-5
9.3.3.6	C – Continue From a Breakpoint	9-5
9.3.3.7	nnnnC – Continue nnnn +1 Times from Breakpoint	9-5
9.3.3.8	D – Open Data Field	9-6
9.3.3.9	F – Open Current Field	9-6
9.3.3.10	M – Open Search Mask	9-6
9.3.3.11	M LINE FEED – Open Lower Search Limit	9-6
9.3.3.12	M LINE FEED LINE FEED – Open Upper Search Limit	9-6
9.3.3.13	nnnnW – Word Search	9-6
9.4	ERRORS	9-7
9.5	PROGRAMMING NOTES	9-7
9.6	ODT COMMAND SUMMARY	9-7
APPENDIX A	**CHARACTER/CONTROL CODES AND SPECIAL SYMBOLS**	**A-1**
APPENDIX B	**USEFUL MATHEMATICAL SUBROUTINES**	**B-1**
B.1	UNSIGNED INTEGER MULTIPLICATION SUBROUTINE	B-2
B.2	UNSIGNED FRACTIONAL MULTIPLICATION SUBROUTINE	B-3
B.3	UNSIGNED INTEGER DIVISION SUBROUTINE	B-4
B.4	UNSIGNED FRACTIONAL DIVISION SUBROUTINE	B-5
APPENDIX C	**USER SERVICE ROUTINE**	**C-1**
C.1	CALLING THE USR	C-1
C.1.1	Standard USR Call	C-1
C.1.2	Direct and Indirect Calling Sequence	C-3
C.2	USR FUNCTIONS	C-3
C.2.1	FETCH Device Handler Function Code = 1	C-3
C.2.2	LOOKUP Permanent File Function Code = 2	C-5
C.2.3	ENTER Output (Tentative) File Function Code = 3	C-6
C.2.4	The CLOSE Function Function Code = 4	C-7
C.2.5	Call Command Decoder (DECODE) Function Code = 5	C-8
C.2.6	CHAIN Function Function Code = 6	C-9
C.2.7	Signal User ERROR Function Code = 7	C-10
C.2.8	Lock USR In Memory (USRIN) Function Code = 10	C-10
C.2.9	Dismiss USR From Memory (USROUT) Function Code = 11	C-11
C.2.10	Ascertain Device Information (INQUIRE) Function Code = 12	C-11
C.2.11	RESET System Tables Function Code = 13	C-12
APPENDIX D	**THE COMMAND DECODER**	**D-1**
D.1	COMMAND DECODER CONVENTIONS	D-1
D.2	COMMAND DECODER ERROR MESSAGES	D-2
D.3	CALLING THE COMMAND DECODER	D-2
D.4	COMMAND DECODER TABLES	D-3
D.4.1	Output Files	D-3
D.4.2	Input Files	D-4

CONTENTS (CONT.)

			Page
	D.4.3	Command Decoder Option Table	D-5
	D.4.4	Example	D-5
	D.5	SPECIAL MODE OF THE COMMAND DECODER	D-7
	D.5.1	Calling the Command Decoder Special Mode	D-7
	D.5.2	Operation of the Command Decoder in Special Mode	D-7
APPENDIX E		**OS/78 DEFAULT FILE NAME EXTENSIONS**	**E-1**
APPENDIX F		**USING DEVICE HANDLERS**	**F-1**
	F.1	CALLING DEVICE HANDLERS	F-1
	F.2	OS/78 DEVICE HANDLERS	F-3
	F.2.1	Line Printer (LPT, LQP)	F-3
	F.2.2	File-structured Devices (SYS, DSK, RXA0, RXA1, RXA2, RXA3)	F-3
	F.2.3	Terminal Handlers (TTY, SLU2, SLU3)	F-4
	F.2.4	Multiple Input Files	F-4
APPENDIX G		**OS/78 ERROR MESSAGE SUMMARY**	**G-1**
	G-1	SYSTEM HALTS	G-1
	G-2	ERROR MESSAGES	G-2
GLOSSARY			**Glossary-1**
INDEX			**Index-1**

FIGURES

FIGURE	6-1	BASIC Editor Commands and Related Uses	6-3
	7-1	Creating a FORTRAN IV Program	7-1
	7-2	Executing a FORTRAN IV Program	7-2
	7-3	Compiling a FORTRAN IV Program	7-3
	7-4	Loading and Executing a FORTRAN IV Program	7-3
	7-5	Main and Subprogram for Calculating the Volume of a Regular Polyhedron	7-25
	7-6	Nested DO Loops	7-40

TABLES

TABLE	2-1	OS/78 Assumed Extensions	2-5
	2-2	CCL Options	2-9
	2-3	OS/78 Command Summary	2-15
	3-1	COMPARE Options	3-6
	3-2	COMPARE Error Messages	3-6
	3-3	COPY Options	3-11
	3-4	COPY Error Messages	3-13
	3-5	CREF Options	3-15

TABLES (CONT.)

		Page
3-6	CREF Error Messages	3-16
3-7	DELETE Options	3-20
3-8	DELETE Error Messages	3-20
3-9	DIRECT Options	3-23
3-10	DIRECT Error Messages	3-23
3-11	DUPLICATE Options	3-25
3-12	DUPLICATE Error Messages	3-25
3-13	HELP Error Messages	3-30
3-14	Available OS/78 Help Files	3-31
3-15	LIST Options	3-33
3-16	LIST Error Messages	3-33
3-17	MAP Options	3-38
3-18	MAP Error Messages	3-38
3-19	RENAME Options	3-43
3-20	Job Status Word	3-45
3-21	SET Command Attributes	3-48
3-22	SET Error Messages	3-52
3-23	SUBMIT Options	3-56
3-24	TYPE Options	3-58
3-25	TYPE Error Messages	3-59
4-1	Editor Options	4-2
4-2	Editor Key Commands	4-3
4-3	Editor Error Codes	4-19
4-4	Editor Command and Special Characters	4-19
5-1	PAL8 Options	5-8
5-2	Use of Arithmetic Operators	5-16
5-3	PAL8 Error Codes	5-32
6-1	Decimal/Character Conversions	6-46
7-1	Standard FORTRAN IV File Extensions	7-4
7-2	OS/78 FORTRAN IV Operational Process	7-5
7-3	FORTRAN IV Compiler Run-Time Options	7-6
7-4	Compiler Error Messages	7-7
7-5	Loader Run-Time Options	7-9
7-6	Loader Error Messages	7-10
7-7	Run-Time System Options	7-14
7-8	Run-Time System Error Messages	7-15
7-9	Logical Operators and Their Meanings	7-32
7-10	Relational Operators and Their Meanings	7-32
7-11	Truth Table for Logical Expressions	7-34
7-12	Conversion Rules for Assignment Statements	7-35
7-13	Numeric Field Codes	7-45
7-14	Conversion Under G Format	7-46
7-15	Device Control Statements	7-54
7-16	FORTRAN IV Statement Summary	7-63
8-1	BATCH Run-Time Options	8-2
8-2	BATCH Monitor Commands	8-3
8-3	BATCH Error Messages	8-6
9-1	ODT Command Summary	9-8
C-1	Summary of USR Functions	C-2

PREFACE

This manual describes the OS/78 Operating System for the DECstation 78 Minicomputer system.

Chapter 1 briefly describes the system and the contents of the software provided on the master diskettes. The basic instructions on how to use the OS/78 Operating System are contained in Chapter 2. This chapter should be read in its entirety prior to using OS/78 to gain familiarity with the operating characteristics and features of the system. The commands that direct computer operations are given in alphabetical order in Chapter 3.

The OS/78 Editor, a program that provides the facility for creating and modifying ASCII source files, is described in Chapter 4. PAL8, a symbolic assembler, is described in Chapter 5. The two high-level programming languages, BASIC and FORTRAN IV, that run under OS/78 are described in Chapters 6 and 7, respectively. Chapter 8 describes BATCH, and Chapter 9 contains information on changing and correcting programs using ODT (Octal Debugging Technique).

For advanced users, information on the use of device handlers, the User Service Routine, and the Command Decoder is contained in the appendices. Also included in the appendices are useful mathematical multiplication and division subroutines as well as an alphabetical summary of all OS/78 error messages.

Related handbooks and manuals that will be helpful to the OS/78 user are as follows:

Introduction to Programming (DEC-08-XINPA-A-D)
OS/8 Handbook (DEC-S8-OSHBA-A-D)
OS/8 Software Support Manual (DEC-S8-OSSMB-A-D)
DECstation/78 User's Guide (EK-VTX-78-TM-OP-001)
DECstation/78 Maintenance Manual (EK-VTX-78-TM-001)

CHAPTER 1
INTRODUCTION TO OS/78

The OS/78 operating system is a practical and reliable operating system designed for the DECstation 78 Minicomputer system. The operating system permits the use of the VT-52 video terminal, dual-drive floppy disk system, either an LA78 or LQP78 line Printer, and all available memory up to 16K. In addition, two auxiliary terminals and a second pair of floppy disks can be supported by the system. OS/78 offers a versatile set of keyboard commands that supervise a comprehensive library of system programs. These programs allow you to develop programs using PAL8 assembly language, BASIC and FORTRAN IV. A BATCH stream processor is also included, providing batch-mode (unattended) system operation. This chapter briefly describes both the DECstation 78 Minicomputer and the OS/78 software.

1.1 DECSTATION 78 MINICOMPUTER SYSTEM

The DECstation 78 Minicomputer system consists of four units:

1. a terminal,
2. the dual disk drives,
3. the central processor, and
4. the main memory.

In addition, a line printer (either an LA78 or LQP78), a second pair of disk drives, and up to two auxiliary terminals can be supported, through the serial asynchronous interface ports. These ports are suitable for primary and secondary communications applications, terminals, and the attachment of a variety of devices.

The console terminal provides the means for interacting with or "talking" to the computer. It serves as the primary source of OS/78 command inputs.

The dual-disk drives include two disk drives that allow the use of flexible disks called diskettes or floppy disks. The diskettes perform two functions:

1. The OS/78 software that allows the computer system to be run is prerecorded on them.
2. They provide space to store programs and files that have been created.

The central processor unit (CPU) is located within the terminal. It is the "heart" of the computer system. The CPU executes instructions from main memory and controls all peripheral devices.

The main memory is the main storage area in the computer from which instructions are fetched and executed, and in which data may be stored and manipulated.

For more information on the DECstation 78 Minicomputer System, refer to the *DECstation 78 User's Guide.*

1.2 OS/78 SOFTWARE

OS/78 can run programs written in three languages: PAL8 Assembly Language, BASIC, and FORTRAN IV. The manipulation of program source and data files is done by typing in the OS/78 commands at the terminal. The software to do this is provided on the two master diskettes. Only one of the two diskettes is used (in the left-hand drive) as a system device. This leaves the right-hand drive free for using blank diskettes to store programs and data.

Diskette 1 contains the following Commonly Used System Programs (CUSPS) or Program Names:

ABSLDR.SV	CCL.SV
DIRECT.SV	PIP.SV
PAL8.SV	SRCCOM.SV
CREF.SV	RXCOPY.SV
FOTP.SV	BATCH.SV
EDIT.SV	BASIC.SV
BITMAP.SV	BLOAD.SV
SET.SV	BASIC.AF
BCOMP.SV	BASIC.FF
BRTS.SV	
BASIC.SF	
RESEQ.BA	
HELP.SV	
HELP.HL	

In addition, there are several demonstration programs on Diskette 1 that begin with the name "DEMO". These programs are discussed in Section 2.3 of Chapter 2.

Diskette 2 contains the following Commonly Used System Programs (CUSPS) or Program Names:

PIP.SV	CCL.SV
DIRECT.SV	EDIT.SV
FOTP.SV	F4.SV
BATCH.SV	PASS2O.SV
PASS2.SV	RALF.SV
PASS3.SV	FRTS.SV
LOAD.SV	RXCOPY.SV
FORLIB.RL	SET.SV
HELP.SV	ABSLDR.SV
HELP.HL	

Essentially, the PAL8 assembler, BASIC, and utility programs are located on Diskette 1, while the FORTRAN IV system programs are located on Diskette 2. It is not necessary to know the meaning of these program names since they are automatically called up when required by OS/78.

CHAPTER 2
GETTING ON THE AIR

This chapter provides the basic information needed to use the OS/78 operating system. You should read the entire chapter prior to using the system to become familiar with the features and capabilities of OS/78.

2.1 STARTING THE SYSTEM

2.1.1 Loading the Diskettes

The diskettes that contain the system programs are loaded by simply sliding them, label side up, into the open drive slots. The system diskette is loaded into Drive 0, which is the left-hand slot.

After insertion, close the door; the latch will "click" when the door is closed properly. Open the doors by pinching the door latch between your thumb and index finger.

Since the diskette is thin and flexible, exercise caution when handling and storing it. Hold it by the edges and do not touch the exposed portions that show through the protective cover. Also, use only felt-tip pens when writing on diskette labels. Store them in a dust-free area, and do not expose them to magnetic devices, direct sunlight, heat, or humidity.

NOTE

The system diskette may be placed in Drive 1 and used there, if necessary.

2.1.2 Calling the Keyboard Monitor

Once a system diskette is properly loaded into Drive 0, the OS/78 Monitor is called by pressing the START button on the right-hand side of the console. After a brief pause to read in the OS/78 Monitor, an OS/78 command summary is displayed on the terminal. The system will then display a dot (.) on the terminal screen. The dot means that OS/78 is ready to accept a command. You can now type any OS/78 command.

2.1.3 Using the Keyboard

The keyboard on the terminal allows data to be entered into the computer. Most of the keyboard is identical to a typewriter keyboard. Unlike a typewriter, pressing a key does not automatically display a character on the keyboard screen. Pressing a key causes the character selected to be sent to the Central Processing Unit (CPU). The program running would normally then transmit this character back to the terminal, thereby displaying it. Consequently, typing a character will have no effect if OS/78 is not running, or if OS/78 is busy running a program that is not waiting for input from the keyboard.

Certain keys have special functions when used in OS/78 commands. They are as follows:

RETURN key

This key is used to terminate OS/78 command lines, Editor command lines, program statements, and responses to program input queries.

DELETE key

This key is used to correct typing errors made while entering Editor command lines, Monitor command lines, and program statements.

LINE FEED key

This key is used while entering Monitor command lines. It causes the Monitor to display the command line as it has been entered so far as a convenience check.

CTRL keys

This key is always used in combination with another key. The CTRL key is held down while the other key is pressed. In this manual, these combinations are shown by using a slash (/) between CTRL and the other key such as CTRL/U, CTRL/C. The effect of these combinations is as follows:

CTRL/C	Terminates execution of the currently active program, if any, and returns control to the Monitor. It also returns control from a BASIC program to the BASIC editor.
CTRL/O	Stops the display of characters on the screen.
CTRL/S	Suspends terminal output of a program or monitor command but does not terminate the program or command.
CTRL/Q	Resumes the terminal output that was suspended by CTRL/S.
CTRL/U	Deletes a line typed to the Monitor or Editor.

The CTRL key commands require no terminator; the system performs the function as soon as you type the command.

ESCape key

This key performs certain functions when using the OS/78 Editor and running FORTRAN IV and PAL8 programs. The function of this key is described in those sections where its use is applicable.

2.1.4 Typing Convention and Symbology

One convention will be used throughout this manual to indicate what the system will display and what you should type. Anything that is not underlined must be entered (typed-in) through the keyboard. Everything that the system displays will be underlined. For example:

```
.DATE
NONE
.
```

In this example, the system displays the monitor dot, you type DATE (and then press the RETURN key). The system displays NONE, and returns to the MOnitor dot (.).

The following keyboard characters will be designated by symbols to indicate specific operations that should be performed when using OS/78. They are as follows:

Key	Symbol	Function
LINE FEED	(LF)	Perform a line feed
TAB	(TAB)	Skip up to eight spaces
ESC	(ESC)	Performs special functions when using PAL8, FORTRAN and the OS/78 Editor
DELETE	(DEL)	Delete a character

NOTE

In this manual all command and input lines are followed by a carriage return (pressing the RETURN key) unless otherwise specified by using one of the above symbols.

2.2 MAKING A BACKUP COPY

It is important that a backup copy be made of the master diskettes.

NOTE

The master diskettes should never be used for running the system or altered in any way.

Make a backup copy by using the following procedure.

1. Place the master diskette in Drive 0, the left-hand slot, and a blank diskette in Drive 1.
2. Start the system by pressing the START button.
3. In response to the monitor dot (.), type the DUPLICATE command followed by a carriage return (pressing the RETURN key) as follows:

```
.DUPL RXA1:<RXA0:
```

   The system will duplicate the contents of the diskette in Drive 0 onto the diskette in Drive 1, and when completed, will return to the Monitor dot.
4. Remove both diskettes and label the diskette taken from Drive 1 with the appropriate markings.
5. Repeat Steps 1 through 4 for the second master diskette.
6. Store the master diskettes in a safe place.
7. Choose one of the copies of the two system diskettes and place it into Drive 0. This diskette will be the system diskette.
8. Push the START button and you are ready to use OS/78.

2.3 DEMONSTRATION PROGRAMS

A series of programs are provided on Diskette 1 to demonstrate some system capabilities of OS/78. These programs are run via a self-explanatory batch stream. To initiate this demonstration, start the system and type the following in response to the Monitor's dot (.):

```
.SUBMIT DEMO/H
```

The batch stream controlling the demonstration repeats it every three minutes until a CTRL/C is typed.

When the Monitor dot appears, type

```
.SET TTY ARROW
.SET TTY NO ESC
```

These commands restore OS/78 to its state prior to the demonstration. Once you become familiar with the system, you may wish to remove the files comprising the demonstration to make room for other programs. To do so, type

```
.DEL DEMO??.*
```

2.4 NAMING DEVICES AND FILES

2.4.1 Devices

Each device on the system must be referred to by name. The device names that are recognized by the system are as follows:

DSK	Default output device; usually same as SYS
SYS	The diskette inserted in Drive 0, upon which the system programs reside.
RXA0	The diskette inserted in Drive 0
RXA1	The diskette inserted in Drive 1
RXA2	The diskette inserted in Drive 2
RXA3	The diskette inserted in Drive 3
TTY	Keyboard/screen
LPT	Line printer
LQP	Letter quality printer
SLU2	Serial port #2
SLU3	Serial port #3
BAT	Batch handler

Each device on the system is recognized by the permanent device name given above and which is built into the system. However, a particular name can be given to a device called a user-defined device name by using the ASSIGN command. For example,

```
.ASSIGN RXA1 PLACE
```

causes all future references to PLACE to address the device RXA1. More than one user-defined device name can be active at a time. The DEASSIGN command causes OS/78 to eliminate any user-defined device names. Also, all such user-assigned names are lost when restarting OS/78 with the START button.

2.4.2 File Names and Extensions

When referring to a diskette, the system will usually expect a file to be specified. Files are referenced symbolically by a name of up to six alphanumeric characters followed, optionally, by a period and an extension of up to two alphanumeric characters. The extension to a file name is generally used as an aid for remembering the format of a file. Some commonly used extensions are given in Table 2-1 and Appendix E.

In most cases, conformance should be maintained to the standard file name extensions established for OS/78. If an extension is not specified for an output file, some system programs append assumed extensions. Where an extension for an input file is not specified, the system searches for that file name with the default extension. Failing to find such a file, a search is then done for the original file without an extension. For example, if PROG were specified as an input file to PAL8 the assembler for the OS/78 Operating System, the system would first look for the file PROG.PA (since .PA is the standard extension for PAL8 input files). If PROG.PA were not found, the system would try to find the file PROG (with no extension). As not all system program's utilize default extensions, refer to Table 2-1 and the individual system programs for details.

Some programs also accept the characters * and ? in file names. They are called wild-card characters. These characters have special meanings to the programs involved. The use of these characters is described in detail in Section 2.5.6.

Since files are the basic units of the OS/78 system, an understanding of files and their structure is helpful when using OS/78. A description of files and file-structured devices is given in Section 2.8. This section can be skipped since it is not necessary for properly operating the OS/78 system, but does give an insight into what is happening in the system.

2.5 ENTERING MONITOR COMMANDS

OS/78 operating system commands are entered at the terminal in response to the dot printed by the Monitor. Pressing the RETURN key initiates execution of the entered command. Some commands require a particular format to distinguish between input and output files or to name certain devices. Any errors that are made while utilizing these commands result in an error message being displayed by the Monitor. Error messages are summarized in Appendix G.

Table 2-1 OS/78 Assumed Extensions

Extension	Meaning
.BA	BASIC source file (default extension for a BASIC input file).
.BI	Batch input file (input for BATCH).
.BN	Absolute binary file (default extension for ABSLDR and BITMAP input files; also used as default extension for PAL8 binary output file).
.CM	Command file.
.DI	Directory listing file.
.FT	FORTRAN language source file (default extension for FORTRAN input files).
.LD	FORTRAN load module file (default assumed by run-time system, FORTRAN IV loader).
.LS	Assembly listing output file (default extension for PAL8).
.MP	File containing a loading map (used by the Linking Loader, MAP command).
.PA	PAL8 source file.
.RA	RALF assembly language file (FORTRAN IV).
.RL	Relocatable binary file (default extension for a Linking Loader input file).
.SV	Memory image file (SAVE file); default for the R, RUN, SAVE, and GET commands.
.TM	Temporary file generated by system.

OS/78 commands and the abbreviated forms for each command are summarized at the end of this chapter. Greater detail on each command is given in Chapter 3.

2.5.1 OS/78 Command Format

An OS/78 command is usually made up of the following parts:

1. a command word or words,
2. a list of output devices and files,
3. a left-angle bracket (<),
4. a list of input devices and files,
5. a series of options, and
6. the RETURN or ESCape key.

Parts 2 through 5 above are called the command parameters because they supply information concerning the devices and files that are affected by the command.

The general format for a command may thus be expressed as follows:

.COMMAND dev:outfil.ex<dev:infil.ex/options (terminator)

where COMMAND is any OS/78 command. Both the input and output specifications consist of a device, a file name, and a file name extension. For each input specification given without explicitly naming a device, the device associated with the previous input specification is assumed. If no device has been given for the first input specification, then DSK: is assumed. Output specifications also consist of a device, a file name, and a file name extension. Output specifications are optional. If no output device is given, then DSK: is assumed; if no file name is given, then any or all the files on the designated device are assumed, the exact action depending on the command.

The output specification is to the left of the left-angle bracket (<) and the input specification is to the right of the bracket. Many options can be stipulated with commands to perform various operations in a simpler way; these are discussed in Section 2.5.4. The command line terminator is usually a carriage return (pressing the RETURN key) that causes the system to begin execution of the command. In some cases, the ESCape key is used, and its use will be explained where applicable.

The command word and parameters in each command must obey the following rules:

1. The command word or words may be abbreviated only as indicated in Table 2-3 or in Chapter 3.
2. No command may have more than three output devices or files, and these must be separated by commas. Some commands such as COPY and DIRECT may have no more than one output specification.
3. The left-angle bracket (<) may be omitted if no output devices or files are specified.
4. In general, no command may have more than nine input devices or files, separated by commas. DIRECT and COPY commands are limited to five input devices or files.
5. The file name may not contain more than six characters; the extension may not contain more than two characters.
6. In a command line, the device name is followed by a colon (:), and is always separated from any file name by a colon.

For example, the command line

```
.DIR RXA1:/E
```

when executed by pressing the RETURN key, displays the directory of RXA1 on the terminal screen. The E option also causes the system to display any empty files in the order that they are located on the diskette.

The command line

```
.COPY RXA0:<RXA1:ARITH.PA,MULT.PA,DIVIDE.PA
```

when executed, transfers the specified input files ARITH.PA, MULT.PA and DIVIDE.PA from RXA1 onto RXA0.

Commands also can be simplified by the use of defaults. The use of defaults is explained in Section 2.6.

2.5.2 Incorrect Commands

If you enter an incorrect command line and then press RETURN, one of the following results will occur:

1. If the command is not recognized, the system will respond with a question mark and a dot, and then wait for you to enter a correct command.
2. If the command is recognized, execution will begin immediately. If an error is noted in the command line, press CTRL/C as soon as possible to terminate the action and return control to the Monitor. Depending upon the type of command and the files involved, this may or may not prevent execution.

   Execution of an incorrect command can destroy important files if it occurs when using the following commands:

   COPY SAVE SQUISH DELETE ZERO

In fact, the error may actually destroy part of the software. Therefore, be extremely careful when using these commands. Any specific cautions that should be exercised when using these commands are fully described in the command chapter (Chapter 3). Always be sure that a back-up copy of the system diskette has been made, along with and all other important files that may have been created (Section 2.2).

Section 2.5.3 describes how to correct any errors that have been made in the command line prior to executing the command.

2.5.3 Correcting Errors

Until the RETURN key is pressed, the command line will not be processed by the system. Therefore, the command line is still available for correction. Always check the command lines before pressing the RETURN key since a line with errors may cause undesirable results.

To correct typing mistakes, use the DELETE key. This key erases the last character typed. Successive DELETEs each cause one more character to be erased. The correct character or characters can then be typed.

A command line may be deleted completely before it is entered by typing CTRL/U (produced by holding down the CTRL key and pressing the U key). This echoes as ^U and returns control to the Monitor without processing the current line. The Monitor dot (.) indicates that the system is ready to accept a new command. The command line can then be retyped. Use the LINE FEED key to verify the contents of a command that is in the process of being entered, that is, whatever has been entered in the command line thus far will be displayed.

2.5.4 Using Input/Output Options

In addition to output and input files that are specified in the command line, there are various options that can be typed in to perform certain functions. Options are either numbers or alphanumeric characters. Numbers used as options are generally contained in the command line with the equal sign (=) or square brackets ([]) construction. The alphanumeric option characters are set off from the I/O specifications by the slash (/) character for single character options, and parentheses for a string of single characters. The usage of the slash, parentheses, equal sign, and square brackets is explained below. In addition, a series of switch options known as CCL options are available for use in the command line.

Equal Sign Construction – An equal sign (=) followed by an octal number may occur only once in a command line and must be followed by a separator character (comma or left-angle bracket), other options, or a line terminator (RETURN or ESCape). For example,

```
.DIRECT SYS:=3
```

will list the directory of the diskette in Drive 0 on the screen in three columns.

Slash Construction – A single alphanumeric character is preceded by a slash and can occur anywhere in the command line (even in the middle of a file name) although the usual position is at the end of the line. For example,

```
.COPY RXA1:SECOND.EX<RXA0:FIRST.EX/T
```

means the same as

```
.COPY RXA1:SECOND.EX/T<RXA0:FIRST.EX
```

The option /T instructs the Monitor to assign the current date to the output file.

Parentheses Construction – When two or more letter options are used, they may be grouped together inside parentheses. This construction is also valid anywhere in the command line. For example,

```
.COPY RXA1:OUTPUT.EX<SYS:INPUT.EX(QT)
```

means the same as

```
.COPY RXA1:OUTPUT.EX<SYS:INPUT.EX/Q/T
```

Square Bracket Construction – The square bracket construction can only occur immediately after an output file name and consists of an open square bracket, a decimal number between 1 and 255, and a close square bracket. The square bracket construction is generally only used when necessary to optimize file storage.

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

```
.PAL BINARY[19],LISTING[200]<SOURCE
```

The output files are a file named BINARY on device DSK: having a maximum length of 19 blocks, and a file named LISTIN (only six characters are significant) on the device DSK: with a maximum length of 200 blocks. The input file is SOURCE on device DSK.

CCL Options – This family of options is of the form

-ex

where ex is one of the options specified in Table 2-2.

In the following example, typing

```
.PAL TEST-T
```

will assemble TEST.PA, store the resulting binary program in TEST.BN on DSK, and display the program listing on the terminal.

2.5.5 Remembering Previous Arguments

A feature of the system is that it remembers arguments used by the commands COMPILE, CREATE, LOAD, PAL, EDIT, and EXECUTE, and can use these arguments in other commands. The remembrance for the EDIT command is located in a separate area from the others. Each time these commands are executed, the command with its argument is remembered in a temporary file. Therefore, the file name used last can be recalled for the next command without again specifying the arguments. If, for example, the EXECUTE command

```
.EXECUTE TEST1.PA
```

was entered previously, compilation of TEST.PA would be accomplished by simply typing

```
.COMPILE
```

These commands do not remember arguments typed on previous days.

Table 2-2 CCL Options

Option	Meaning
-L	Send output to LPT.
-LS	Generate a listing file (used with the COMPILE, EXECUTE, and PAL commands). The listing file is written onto SYS: if no output device is specified and is given a .LS extension. The listing file name is the same as the file name that immediately preceded the CCL -LS option.
-MP	Generate a memory map (used with the COMPILE, EXECUTE, and PAL commands).
-NB	Do not create a binary file (used with the COMPILE, EXECUTE, and PAL commands).
-T	Send output to terminal. For example, typing .DIR-L will list the directory of the system device on the device named LPT.
-PA	Selects the PAL8 compiler when the files extension does not determine it (used with the COMPILE and EXECUTE commands).
-FT	Selects the FORTRAN IV compiler when the file extension does not determine it (used with the COMPILE and EXECUTE commands).

2.5.6 Using Wildcards

Certain commands allow wildcards in file name specifications. These commands are COPY, DELETE, DIRECT, LIST, RENAME, and TYPE.

Wildcards allow a file name or the extension in a command to be replaced totally with an asterisk or partially with a question mark to designate certain file names or extensions. The wild characters are particularly useful when doing multiple file transfers. The asterisk is used as a wild field to designate the entire file name or extension. This is illustrated in the following examples:

TEST1.*	All files with the name TEST1 and any extension.
*.BN	All files with a BN extension and any file name.
.	All files (except when used with the DELETE command).

The question mark is used as a wildcard character to designate part of the file name or extension. A question mark is used for each unspecified character that is to be matched. For example, PR?? matches all files beginning with PR that are two to four characters long. Other examples are as follows:

TEST2.B?	All files with the name TEST2 and any extension beginning with B.
TES??.PA	All files with a PA extension and any file name from three to five characters long beginning with TES.
??.??	All files with file names of two characters or less.

The asterisk and the question mark can be specified together in the same command line:

???.*	All files with file names of three characters or less.

A specification may not contain embedded *'s. For example, A*B.* is an illegal specification and will produce the following message:

```
ILLEGAL *
```

If an * or ? is included in a command other than COPY, DELETE, DIRECT, LIST, RENAME, or TYPE, the following message appears:

```
ILLEGAL * OR ?
```

2.5.6.1 Wildcard Input File Specifications – This section describes wildcards when specifying input files.

For example, to display the directory entries for all files on RXA1 with file names from two to four characters long and beginning with PR, type

```
.DIR RXA1:PR??.*
```

To display entries for all files on RXA0 with extensions ending with R, type

```
.DIR RXA0:*.?R
```

The following is a list of other illegal wildcards:

A*.BD ABC.*D	Illegal because the asterisk should replace the entire file name or extension
.BA	Illegal because the file name cannot be omitted; it must be specified in some form.
RXA?:C.BA	Illegal because wildcards may not be used in device names.

2.5.6.2 Wildcard Output File Specifications – This section describes the use of wildcards when specifying output files.

The wildcard asterisk may be used in output files, but the question mark is illegal. If no file name is specified, then *.* is assumed.

For example,

```
.DEL *.BN<*.PA
```

is legal. This command deletes any file with a .BN extension if the file and a .PA extension exist.

Another example would be

```
.COPY RXA1:*.BK<SYS:*.PA
```

where all files with a .PA extension would be copied from the system device onto RXA1, having the same file name but given a .BK extension.

The question mark may not be used in output file entries. For example,

```
.COPY RXA1:NEW.??<OLD.BA
```

is not allowed.

Greater detail on using wildcards is given in the *OS/8 Handbook.*

Wildcards, when used with the COPY and DELETE commands, is an extremely powerful feature, but is capable of destroying needed files if not used carefully. Therefore, the following two steps are recommended:

1. Keep a backup copy of the system diskette and all other important files
2. Use the /Q option when running COPY or DELETE with wildcards. This allows you to decide what to do with each file involved before it is acted upon by the system. Type Y to allow the operation, N to disallow it.

2.5.7 Indirect Commands (@ Construction)

When many file names and options are to be included in a single command, they can be put into a file and thus need not be typed each time they are required. This is done by using the @ file construction, which may appear anywhere within the argument (file and option list) portion of a command. The @ construction is of the form

@dev:file.ex

If dev: is omitted, DSK: is assumed. The word file must be a file name. If the extension is omitted, .CM is assumed.

Indirect command files are not permitted in the following commands:

ASSIGN RUN
DEASSIGN SAVE
GET ODT
START DATE
R

The information in the specified file is then put into the command string to replace the characters @dev:file.ex. For example, if a file FLIST.CM is created using the Editor, and contains the string

FILEB,FILEC/L,FILED

then the command

```
.COMPILE FILEA,FILEB,FILEC/L,FILED,FILEZ
```

could be replaced by the command

```
.COMPILE FILEA,@FLIST,FILEZ
```

Carriage returns and line feeds within the file are ignored, but nulls are not; the nulls signify end-of-line. Command files may not exceed one block in length. If a command line is more than 512 characters in length, the following message is printed:

```
COMMAND LINE OVERFLOW
```

The construction @dev:file.ex may not be immediately followed by a letter or digit (it would be absorbed as part of the .ex). The @dev:file.ex may be followed by an apostrophe (') which would then be ignored.

2.5.8 Asterisk (*)

Under certain conditions, OS/78 may display an asterisk. This is a function of a special program called Command Decoder, and signifies a request to enter another list of files for the current system command. Except in cases where its application is specifically described, just type CTRL/C in response to the asterisk to return to the OS/78 Monitor. The Command Decoder is described in Appendix D.

2.6 USING DEFAULTS

The amount of typing necessary for entering commands may be reduced by taking advantage of the default ability of the system. A default is a command parameter that when not typed is assumed by the system. Defaults are generally used to avoid typing device names.

1. The general default device assumed by the Monitor is DSK: (usually equivalent to SYS: or RXA0:), the diskette in Drive 0.
2. For all output files and input files, the default device is DSK:. For example, the following two file descriptions mean the same to the Monitor:

   Command FILEA.AB,FILEB.CD<FILEC.EF
   Command DSK:FILEA.AB,DSK:FILEB.CD<DSK:FILEC.EF

3. When a device name is explicitly given for an input file, any additional input files will default to the last explicit device name:

   Command TTY:<FILEA, FILEB, RXA1:FILEC, FILED
   Command TTY:<DSK:FILEA,DSK:FILEB,RXA1:FILEC,RXA1:FILED

   In this description,

FILEA	was assumed to be on DSK: because it was the first input file.
FILEB	was assumed to be on DSK: because the preceding file (FILEA) was on DSK: (same as RXA0:).
FILEC	was stated to be on RXA1:.
FILED	was assumed to be on RXA1: because the preceding file (FILEC) was on RXA1:.

4. Some of the individual commands assume special default devices. The DIRECT and TYPE commands assume the terminal as the output device. The following command line will display the directory of the diskette inserted in Drive 1 on the terminal screen.

   ```
   .DIRECT RXA1:
   ```

   Note that the left-angle bracket is not needed in the above command line because no output files are specified. However, commands such as CREF (cross reference program) and LIST default to the LPT. Thus, if a LPT is not available, the output of the CREF command must be sent to a file or to the terminal (using the CCL option-T). Specific cases and use of options are discussed in Chapter 3.

2.7 WHERE TO GET MORE INFORMATION

Many times more information is desired such as the function of certain commands, their format, and the particular options that can be used with them. A file named HELP.HL is available to provide this information. The information will be displayed on the terminal by using the HELP command and giving the command name for which information is desired as an argument. For example,

```
.HELP COMPARE
```

will display the following information on the terminal:

```
        SRCCOM.SV


    CALLING COMMANDS:
    .COMPAR DEV:OUTFILE.PA<DEV:INFILE1.PA,DEV:INFILE2.PA
    .COMPAR OUTFILE.PA<INFILE1.PA,INFILE2.PA              /FILES ON DSK

    SWITCHES:
    /B         COMPARE BLANK LINES
    /C         DON'T COMPARE (SLASHED) COMMENTS
    /S         DON'T COMPARE TABS AND SPACES
    /T         CONVERT TABS TO SPACES ON OUTPUT
    /X         DON'T COMPARE OR PRINT COMMENTS

    ERRORS:
    ?0         INSUFFICIENT CORE
    ?1         INPUT ERROR FILE 1 (OR LESS THAN 2 INPUT FILES)
    ?2         INPUT ERROR FILE 2
    ?3         OUTPUT FILE TOO LARGE
    ?4         OUTPUT ERROR
    ?5         CAN'T OPEN OUTPUT FILE
```

If a line printer is attached to the system you can get a hard copy printout by outputting the desired HELP text to the line printer as follows:

```
    .HELP LPT:<PAL,COMPARE
```

or (by using the CCL option -L)

```
    .HELP PAL,COMPARE-L
```

The HELP files for PAL and COMPARE will be printed on the line printer.

2.8 FILES AND FILE-STRUCTURED DEVICES

Files are basic units of the OS/78 system, and an understanding of file structure is helpful for its use. A file is any collection of data to be treated as a unit. The format of this data is unimportant; for example, OS/78 can manipulate several standard formats, including ASCII files, binary files, and memory image files. The important consideration is that the data forms a single unit within the system.

2.8.1 File-Structured Devices

A file-structured device is one that can be logically divided into a number of 256-word blocks and has the ability to read and write from any desired block. Diskettes are file-structured devices but a terminal is not. Currently, a diskette is the only file-structured device supported by OS/78.

All OS/78 file-structured devices must be logically divided into these 256-word blocks. Hence, 256 words is considered the standard OS/78 block size. A given file consists of one or more sequential blocks of 256 words each (consecutively numbered). A minimum of one block per file is required, although a single file could occupy all of the blocks on a device.

OS/78 blocks are conventionally given in octal. File lengths (number of blocks per file) are conventionally given in decimal.

2.8.2 File Directories

To maintain records of the files on a device, blocks 1 through 6 are reserved for the file directory of that device. Thus, file structured devices are also called directory devices. Block zero is unused, except on the system device, in which case it contains the system bootstrap, a program that reads in the OS/78 Monitor.

Entries in this directory inform the system of the name, size, and location of each file, including all empty files and the tentative output file, if one exists. Six blocks are always allocated, though all are not necessarily active at any given time.

2.8.3 File Types

Three types of files exist in the OS/78 system:

1. Empty File
   An empty file is a contiguous area of unused blocks. Empty files are created when permanent files are deleted, or when the ZERO command is used.
2. Tentative File
   A tentative file is a file that is open to accept output and has not yet been closed. Only one tentative file can be open on any single device at one time. Tentative files never appear in directory listings.
3. Permanent File
   A permanent file is a file that has been given a fixed size and is no longer expandable. A tentative file becomes permanent when it is closed.

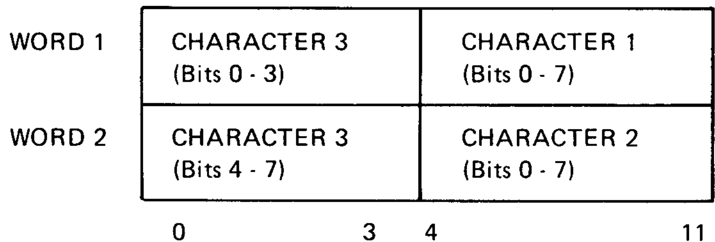
To further understand file types, consider what occurs when a file is created. Normally, in creating a tentative file, the system first locates the largest empty file available and creates a tentative file in that space. This establishes the maximum space into which the file can expand. The user program then writes data into the tentative file. At the end of the data, the program tells the system to close the tentative file, making it a permanent file. When the file is closed, whatever space remains at the end of the tentative file is allocated to a new, smaller, empty file.

2.8.4 File Formats

OS/78 and relevant system programs use three standard file formats:

1. ASCII and Binary Files

ASCII files are packed as three 8-bit characters into two words as follows:



2. Memory Image (.SV Format) Files

A memory image file consists of a header block followed by the actual memory image. The header block is traditionally called the Core Control Block (CCB).

The CCB uses only the first 128 words of the 256 word block reserved for that purpose.

The CCB is a table of information containing the program's starting address, areas of memory into which the program is loaded, and the program's Job Status Word. (Refer to Section 3.26 for a description of the Job Status Word.) The CCB is created at the time the program is loaded into memory and is written onto the .SV file by the SAVE operation.

When a program is loaded, the Starting Address and Job Status Word are read from the CCB and stored in memory. The CCB itself is saved in words 200 (octal) through 377 (octal) of block 37 on the system device. It is placed there by using the GET and RUN commands.

2.9 OS/78 COMMAND SUMMARY

A summary of all OS/78 commands is given in Table 2-3. The table shows the abbreviated form of the command acceptable to the system, command format, and a brief functional description of each command.

Table 2-3 OS/78 Command Summary

Command	Abbreviated	Format	Function
ASSIGN	AS	.AS permdev user-dev-name	Makes a new user-defined device name equivalent to the permanent device name.
COMPARE	COMPA	.COMPA dev:output.ex< dev:file1.ex,dev:file2.ex	Compare two source files line by line and display all their differences.
COMPILE	COM	.COM file.ex <file.ex/options or for multiple input files .COM A,B<C,D	Produce binary files and/or compilation listings for specified program files. This command will chain to either the PAL8 assembler or BASIC compilers dependent upon the filename extension. Under FORTRAN IV, only one input file can be specified.
COPY	COP	.COPY dev:file.ex <dev:file.ex,...	Transfer files from one OS/78 device to another. Up to five input files can be specified.
CREATE	CREA	.CREA file.ex	Open a new file for editing.
CREF	CREF	.CREF file.ex	Assemble and produce a cross-reference listing from the source file. Defaults to LPT, thus the -T option must be used to output to terminal.
DATE	DA	.DA dd-mmm-yy	Enter the date into the system. If no argument is given, print current date on terminal or NONE if no date was specified.
DEASSIGN	DE	.DE	Eliminates all previous user-defined names.
DELETE	DEL	.DEL dev:file.ex,.../options	Delete one or more files. Up to five input files can be specified, separated by commas. All files must reside on the same device.
DIRECT	DIR	.DIR dev:/options	Produce a listing of an OS/78 device directory.

Continued on next page

Table 2-3 (Cont.) OS/78 Command Summary

Command	Abbreviated	Format	Function
DUPLICATE	DUPL	.DUPL outdev:<indev:/options	Reproduce the contents of the input device onto the output device (floppies only).
EDIT	ED	.ED file or .ED file<file	Open an already existing file for editing. Using the input/output construction allows a copy of the original file to be retained, while naming a new output file.
EXECUTE	EXE	.EXE file.ex,file.ex/options or .EXE file.ex<file.ex	Produce binary files and/or compilation listings for the specified program files, load the binary file, and execute the program. Using the input/output format executes the output file while producing from the input file. Under FORTRAN IV, only one file can be specified.
GET	GE	.GE dev:file.ex	Load memory image files (.SV format) into memory from a device.
HELP	HELP	.HELP command	Display instructional information on the specified OS/78 command.
LIST	LI	.LI dev:file.ex,...	List the contents of the specified files on LPT.
LOAD	LO	.LO dev:file.ex	Run one of the OS/78 loaders for either PAL8, BASIC or FORTRAN IV depending upon the extension of the filename.
MAP	MAP	.MAP dev:file.ex<dev:file.ex	Produce a memory map of the specified input file.
MEMORY	MEM	.MEM n	Display the number of fields available or actually being used by OS/78.
ODT	OD	.OD	Load and start the ODT (Octal Debugging Technique) system.
PAL	PAL	.PAL dev:file.ex<dev:file.ex	Run the PAL8 assembler and assemble the specified source file.
R	R	.R file.ex	Load memory image files from the system device (making it equivalent to a RUN SYS file.ex command except that the system scratch area is not affected) and start it at the starting address. If .ex is omitted, .SV is assumed.

Continued on next page

Table 2-3 (Cont.) OS/78 Command Summary

Command	Abbreviated	Format	Function
RENAME	REN	.REN dev:new.ex<dev:old.ex	Change the name of the file from the input name to the output name.
RUN	RU	.RU dev:file.ex	Load a memory image file, move its Core Control Block to the system scratch area, and start the program at its starting address. This command is equivalent to a GET and START command, and must be used to "run" a program that does not reside on SYS:. The colon after dev in the command line may optionally be replaced by a space.
SAVE	SA	.SA dev:file.ex	Save an image of the program currently in memory on the device specified. If .ex is omitted, .SV is assumed.
SET	SET	.SET dev[no] attribute(argument)	Modify the handler for either the line printer or terminal by the attributes or arguments specified in the command line.
SQUISH	SQ	.SQ dev:<dev:	Eliminates all embedded empty files on the device.
START	ST	.ST fnnnn or .ST	Start the program currently in memory at location nnnn in field f. Start the program currently in memory at the starting address specified in the Core Control Block.
SUBMIT	SU	.SU dev:file.ex<dev:file.ex	Run the BATCH program where the output is the optional spooling file and the input is the BATCH input file.
TERMINATE	TER	.TERMINATE	Cause the VT-78 to enter the VT-52 emulator mode.
TYPE	TY	.TY dev:file.ex,...	Display the specified files on the terminal.
UA	UA	.UA (argument)	Remember the argument where the argument must be a legal OS/78 command. UA with no argument executes the last remembered argument.

Continued on next page

Table 2-3 (Cont.) OS/78 Command Summary

Command	Abbreviated	Format	Function
UB	UB	.UB (argument)	Similar to UA
UC	UC	.UC (argument)	Similar to UA
ZERO	ZERO	.ZERO dev:	Zero the specified device directory, deleting any files that may exist on the device.

CHAPTER 3
OS/78 COMMANDS

OS/78 system software supports a powerful set of commands that allows you to direct computer operations. Each OS/78 command has a standard format that should be well-understood prior to using the command.

The format for each command is shown, including one or more examples. Each command description is followed, when applicable, by the options that can be used with the command, and the error messages that could result where using the command.

In response to the dot produced by the OS/78 Monitor, type the command line and those options that can be used with the command to perform a specific function. Press the RETURN key to enter the command into the system for execution. Characters that are underlined are system responses; characters that you type are not underlined.

Depending upon the command, arguments may or may not be specified. Many of the commands have default options that are automatically inserted in the command line by the system if an argument is not explicitly specified. Default options may also vary with each command. The use of defaults is described in Chapter 2, Section 2.6.

3.1 ASSIGN COMMAND

The ASSIGN command assigns a user-defined name to one of the available permanent devices such as SYS or TTY.

Format:

.ASSIGN permdev user-dev-name

Example:

```
.AS RXA1 DEV2
```

In this example, device RXA1 is assigned the symbolic name DEV2 and can now be addressed by both RXA1: and DEV2:. Note that the colon is not used with both the permanent device name and the user-defined name in the ASSIGN command. However, when referenced in most other commands the colons must be used with the user-defined name.

If a user-defined name is not specified in the command line, any one that may have existed is removed and only the permanent device name is valid.

Example:

```
.AS RXA1 DEV2      – Previous ASSIGN command
.AS RXA1           – User-defined name becomes invalid
```

3.1.1 User-Defined Name

The user-defined name that is assigned to a permanent device can be from one to four characters long. If a three or four character name is used it becomes a single word when translated into internal representation. As a result, it is possible that this code could be identical to some other internal code used by the system.

The ASSIGN command is particularly useful if an LQP line printer is attached to the system instead of an LA78 line printer. Since several of the OS/78 commands automatically send output to the LA78 line printer (LPT), assigning LPT to LQP allows output to be sent to the LQP78 line printer.

Example:

```
.AS LQP LPT
```

ASSIGN processing is done entirely by the Monitor.

3.2 BASIC COMMAND

The BASIC command requests execution of the BASIC editor.

Format:

```
.BASIC
```

Example:

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

For additional information on BASIC, refer to Chapter 6.

This command runs CCL.SV and BASIC.SV.

3.3 COMPARE COMMAND

The COMPARE command compares one source input file to another source input file to see if they are identical or different. Differences are sent to the output file.

Format:

```
.COMPARE RXA1:DIFVER<SYS:VERS1.FT,VERS2.FT
```

Example:

```
.COMPA   RXA1:DIFVER<SYS:VERS1.FT,VERS2.FT
```

In this example, every source line in files VERS1.FT and VERS2.FT are compared and the results of the comparison are sent to the file named DIFVER on device RXA1. The file extension .LS is automatically added to DIFVER.

Along with the necessary arguments for the command line are COMPARE options that can restrict or enhance the result of the COMPARE operation. The COMPARE program is commonly used to print all the editing changes between two different versions of a program. This makes it a useful documentation and debugging tool.

Before the first two lines of input are compared, the current version number of the COMPARE command is sent to the output device followed by a header for each input file. The header is the first line in the file which is usually the title, date, file name, or description of that file.

Example:

```
SRCCOM V4A


1)   /THIS IS THE FIRST SOURCE COMPARE FILE
2)   /THIS THE SECOND SOURCE COMPARE FILE
```

When the COMPARE command is executed, lines from the first file that are either identical or different with lines from the second file are sent to the output device until there are three consecutive lines that agree. The first line sent to the output device is preceded by the numeral one, a closing parenthesis, and a 3-digit decimal number. The numeral one represents the first file specified in the command line and the 3-digit decimal number is the page that the proceeding difference group is located on. Each succeeding line is also preceded by the numeral one and closing parenthesis. Out of the three lines that agree, only the first line is sent to the output device which is followed by a group of asterisks on the next line. This ends the first difference group for file 1.

Example:

```
1)002     C
1)        D
1)        E
1)        F
1)        G
1)        H
1)        I
1)        J
1)        K
1)        L
1)        M
1)        N
****
```

The COMPARE command then sends all lines from the second file that both agree and disagree with lines from the first file to the output device until there are three consecutive lines that agree.

NOTE

To change the number of consecutive lines that interrupt processing, use the =k option. By specifying the =k option, any octal number can be used to set the number of consecutive lines that separates one difference group from another.

The first line sent to the output device is preceded by the numeral two, a closing parenthesis, and a 3-digit decimal number. The numeral two represents the second file specified in the command line and the 3-digit decimal number is the page the proceeding difference group is located on. Each succeeding line is also preceded by the numeral two and closing parenthesis. Out of the three lines that agree, only the first line is sent to the output device which is followed by a group of asterisks on the next line. This ends the first difference group for file 2.

Example:

```
2)002    4
2)       6
2)       2
2)       X
2)       B
2)       Y
2)       S
2)       F
2)       M
2)       A
2)       R
2)       N
********
```

If all the lines on both files agree (disregarding the header lines), a message "NO DIFFERENCES" is sent to the output device.

3.3.1 Comparing Part of a Line

The /C option compares all source lines in one file with source lines in another file but disregard all assembly comment lines. In addition to disregarding comment lines in a comparison the output of the comment lines can be restricted by using the /X option. By specifying the /X option, all comments are ignored while the comparison takes place; furthermore, when /X is specified comments are not sent to the output device.

If the format of one file is different from that of another because of tabs and spaces, and a comparison is needed, use the /S option. By specifying the /S option, the COMPARE command compares all source lines but disregards all tabs and differing spaces.

3.3.2 Making Changes While Using the COMPARE Command

Normally, blank lines are disregarded when comparing two files. If a blank line is to be considered as a valid input line instead of a carriage return in a comparison, use the /B option. By specifying the /B option, a blank line is considered to be a valid line containing all blanks.

NOTE

When the /S and /B options are used together, a blank line is not taken into consideration.

This command runs CCL.SV and SRCCOM.SV.

Table 3-1 COMPARE Options

/C	(comment fields)	Ignore all comment fields during the comparison (assumes assembly language source files are being compared)
/B	(blank lines)	Consider a blank line as a valid input line containing blanks instead of a carriage return.
/S	(tabs and spaces)	Ignore all tabs and spaces during the comparison.
/T	(tabs to spaces)	Convert all tabs from the input file to spaces on the output device.
/X	(output of comment fields)	Ignore all comment fields during the comparison and do not send any comments to the output device.

Table 3-2 COMPARE Error Messages

?0	The differences between the two files are too large for an effective compare. NOTE: If the large number of differences is caused by comment lines, use the /X option to alleviate this problem.
?1	There is an input error on file 1, or two input files are not specified in the command line.
?2	There is an input error on file 2.
?3	Ran out of room on the output device.
?4	There is an output error.
?5	An output file cannot be created.

3.4 COMPILE COMMAND

The COMPILE command assembles PAL source input files to produce absolute binary output files, compiles FORTRAN source input files to produce relocatable binary output files and compiles, loads and executes BASIC source input files.

Format:

.COMPILE outdev:file.ex<indev:file.ex/options

Along with the necessary arguments for the command line are several PAL, BASIC, and FORTRAN options that can modify the result of the COMPILE operation. For PAL, BASIC, and FORTRAN options, see the appropriate language chapter.

When the COMPILE command assembles a PAL input file, the absolute binary output file created usually has a .BN extension (the default for PAL8). Whereas, when the COMPILE command compiles a FORTRAN input file, the relocatable binary output file created usually has an .RL extension (the default for FORTRAN IV). Both absolute and relocatable binary output files may be used as input files in the LOAD command.

An absolute binary file is an independent stand-alone program while a relocatable binary file is a dependent program which needs other system programs to aid it in its execution.

The extension of the first input file determines which compiler will be used. The .PA uses PAL8, .FT uses FORTRAN, .BA uses BASIC. If the extension is not specified as part of the file name in the command line, all assembler and compiler extensions are compared with the unspecified extension until a match is made. Then the appropriate assembler or compiler is called and executed. If an extension other than the ones listed in Appendix E is used, specify the correct assembler or compiler extension as a CCL option in the -ex portion of the command line.

Example:

```
.COM RXA1:COMP<ACCTS.03-FT
```

NOTE

If no output file is specified, the first input file specified in the command line is used as the default option. For example,

```
.COMPILE FILE
```

The COMPILE command can recall arguments from a previous COMPILE, LOAD, EXECUTE, or PAL command. This allows the COMPILE command to be typed without any arguments. However, it must be used on the same day because the COMPILE command does not remember arguments typed on a previous day.

NOTE

Refer to the appropriate language chapter for any error messages received as a result of the COMPILE command.

This command causes execution of the CCL.SV program and either PAL8.SV, F4.SV, or BCOMP.SV.

3.5 COPY COMMAND

The COPY command transfers files from one device to another. It also allows you to change the file's name.

Format:

```
.COPY outdev:file.ex<indev:file1.ex...file5.ex/options
```

When the COPY command is executed, all specified input files are copied onto the output device. A message informing you of the files copied followed by the name of each file copied is displayed until copying is done.

Example:

```
.COPY RXA1:<FINE,TEAK,ROSE

FILES COPIED:
TEAK
FINE
ROSE
```

3.5.1 File Transfer

When the COPY command is executed, each input file is copied in its exact format to the output device in the same order they are found on the input device. No changes are made during file transfer and as a result, any file, whether memory image, binary or ASCII, can be transferred. During the transfer, the original creation date is transferred and included in the directory. Along with the necessary arguments for the command line, COPY options can be specified that can change and enhance the result of a file transfer.

If input files are to be transferred to the output device in the exact order specified in the command line, use the /U option.

If the majority of the input files on the input device are to be transferred to the output device, use the /V option and specify the files that are not to be transferred. The COPY command then transfers all the files that are not specified in the command line. Wildcards can be used with the COPY command (described in Section 2.5.6).

3.5.2 Examples of Copy Commands

The following are legal COPY command strings. When the operation is completed, control returns to the Monitor. Some of the examples show the use of wild cards.

Example 1:

```
.COPY RXA1:<MATH.PA
```

This command string transfers file MATH.PA from the device DSK to RXA1.

Example 2:

```
.COPY RXA1:<A,B,C,D,E
```

This command string transfers the files A, B, C, D, and E from the system device to RXA1, producing a list of those files copied.

Example 3:

```
.COPY LPT:<*.FT,*.BA/U
```

This command string lists all FORTRAN files, then all BASIC files on the line printer (same as LIST).

Example 4:

```
.COPY LPT:<RXA0:*.SV,*.BN/V
```

This command string copies from RXA0 to the line printer all files other than memory image (.SV) and binary (.BN).

3.5.3 Predeletion

Before the COPY command copies the designated input files to the output device, the output device is checked for files having the same name as the input files. If an identical file name is found, that file is deleted before file transfer is performed. This process is known as predeletion and, by default, is automatically done each time the COPY command is executed.

If there are no files with the same name on the output device, each file is transferred to the smallest available empty space that it can fit in.

Predeletion is advantageous because it creates space for a new file on the output device by deleting the old file before file transfer. In many cases, the output device has received as many files as it can hold and therefore may not have enough room for a new tentative file. By first deleting the old file, a sufficient amount of space could be created for the new file. Predeletion normally places the new file in the space occupied by the file being replaced. Therefore, if the length of the input file is the same, or shorter than the length of the deleted file, that empty space is filled. Otherwise, the new tentative file is placed somewhere else on the device leaving a gap where the old file was deleted. This alters the order of the files on the output device after transfer.

If predeletion is not desired, use the /N option in the command line. When the /N option is specified, postdeletion takes place.

3.5.4 Postdeletion

If the /N option is used with the COPY command, the designated input files are copied to a tentative file on the output device without checking for files having the same name. When the transfer operation is completed, the tentative file is closed. Closing a tentative file makes it a permanent file and, at the same time, deletes any old files having the same name. This process is known as postdeletion and takes place only when the /N option is specified.

Example:

```
.COPY RXA1:<RXA0:ACCT06/N
```

A file named ACCT06 on device RXA0 is copied to a tentative file on device RXA1. After file transfer is completed, the file is closed and any other file on device RXA1 with a filename of ACCT06 is deleted.

The primary advantage of postdeletion is that if an input/output error (due, for example, to a bad diskette) occurs on reading an input file, the corresponding output file will be preserved instead of deleted. Always use the /N option when copying important files to backup diskettes to avoid losing both file and backup file in the event of an input/output error.

3.5.5 Considering the Date on a Transfer

When files are transferred from one device to another, their creation dates are also transferred. These dates along with their new or original file names are included in the output directory. If it is desired to assign the current date to an output file and disregard the creation date, use the /T option. This option allows the COPY command to transfer files from one device to another, and append the file names and the current date to the output directory.

To transfer all input files that have the current date, use the /C option. When the /C option is used, the COPY command copies only those files that were created or changed on the current date and appends the filenames and their creation dates to the output directory.

However, the transfer of all input files that have a creation date other than the current date is done by using the /O option. By specifying the /O option in the command line, the COPY command copies only those files with a date other than the current date from one device to another, and adds the file names and their creation dates to the output directory.

3.5.6 File Protection During a Transfer Operation

When transferring a large number of files, use the /F (failsafe) option. When the /F option is specified and an input file is encountered which will not fit on the output device, a message is printed on the terminal informing you to dismount the current diskette and mount a new diskette on the same drive. To continue the COPY operation, type any character. If the /F option had not been specified, then files which did not fit on the output device would not get copied. Instead, a warning message would be printed:

NO ROOM, SKIPPING filename

Example

```
.COPY RXA1:<RXA0:*.FT/F




FILES COPIED:
VERS.9.FT
VERS12.FT
VERS10.FT
VERS11.FT
VERS7.FT

MOUNT NEXT OUTPUT VOLUME:

VERS5.FT
VERS6.FT
VERS8.FT
```

When the /F option is used, all output directories on new devices that might be used for continued COPY operations should be emptied (zeroed). This is done by using the ZERO command before the COPY operation begins. It is possible to continue COPY operations to a device with already existing files. However, if the length of the next file to be transferred exceeds the lengths of the empty spaces on the output device, another message for a new device to be mounted is printed on the terminal.

If many files are being transferred at one time, especially ones with similar names, each affected file specified in the command line can be verified as the correct file for the operation by using the /Q (query) option. When the /Q option is specified, the first input file name followed by a question mark (?) is printed on the terminal. At this time, a decision must be made on whether or not it is the correct file for the operation. If yes, type a Y and that file is transferred to the output device. If no, type any other character and that file is ignored. After your file is either transferred or ignored, the next input file name followed by a question mark is printed on the terminal. This questioning continues until all affected files in the command line have been processed.

Example:

```
.COPY SYS:<RXA1:ASSEM.FA,MAIN.FT/Q

FILES COPIED:
MAIN.FT?Y
ASSEM.FA?N
```

In this example, the file named ASSEM is not transferred.

3.5.7 Terminating Copy Operations

The only control character used to abort COPY operations is CTRL/C.

When CTRL/C is typed during a COPY operation, file transfer terminates immediately and control returns to the Monitor. Because file transfer is terminated before a successful completion, some requested files may not have been transferred.

Example:

```
.COPY RXA1:<RXA0:TABLE.FT
^C
FILES COPIED:

.
```

This command causes the execution of batch the CCL.SV and FOTP.SV programs.

Table 3-3 COPY Options

Option	Meaning
/C (current date)	Transfer all input files that have the current date to the output device.
/F (failsafe)	A file will not fit on the output device. The message: MOUNT NEW DEVICE: is printed on the terminal. Dismount current device and mount new device on same unit. Continue the COPY operation by typing any character. The new device must have a valid OS/78 directory on it.
/T (today's date)	Transfer all specified input files to the output device and at the same time, change the creation date on the output files to the today's date.
/U (in exact order)	Find and transfer the designated input files in the exact order they are specified in the command line, not the order they are found on the input device.
/V (other than the specified files)	Transfer all files on the input device that are not specified in the command line to the output device.

Continued on next page

Table 3-3 (Cont.) COPY Options

Option	Meaning
/W (current version number)	Before any specified input files are transferred to the output device, print the current version number of the program used by the COPY command.
/N (no predeletion)	After each file is transferred as a tentative file, any existing file having the same name is deleted and the tentative file becomes permanent. This is postdeletion.
/O (other than the current date)	Transfer only those input files with dates other than the current date.
/Q (query the user)	Each time a file name followed by a question mark is printed on the terminal, type a Y (yes) to transfer the file or any other character (no) if the file is not to be transferred.

NOTE

All output directories on devices that could be used for continued COPY operations (/F) should have valid OS/78 directories on them. One way to accomplish this is to ZERO a diskette before execution of the COPY operation. This diskette can then be used as an overflow diskette for any files that do not fit on the original diskette. (See the ZERO command).

Table 3-4 COPY Error Messages

Message	Meaning
BAD INPUT DIRECTORY	The directory on the specified input device does not exist or is not valid.
BAD OUTPUT DEVICE	This message appears when a non-file structured device is specified as the output device.
BAD OUTPUT DIRECTORY	The directory on the specified output device does not exist or is not valid.
ERROR ON INPUT DEVICE SKIPPING (file name)	The file specified is not transferred, due to a disk-reading error, but any previous or subsequent files are transferred and indicated in the new directory.
ERROR ON OUTPUT DEVICE SKIPPING (file name)	The file specified is not transferred, but any previous or subsequent files are transferred and indicated in the new directory.
ERROR READING INPUT DIRECTORY	A disk-reading error occurred on the input device.
ERROR READING OUTPUT DIRECTORY	A disk-reading error occurred on the output device.
ERROR WRITING OUTPUT DIRECTORY	A disk-reading error occurred on the output device.
ILLEGAL *	An * was entered as an embedded character in a file name, e.g., TMP*,BN (see Section 2.5.6 on wild cards).
ILLEGAL ?	A ? was entered in an output specification (see Section 2.5.6 on wild cards).
NO FILES OF THE FORM xxxx	No files of the form (xxxx) specified were found on the current input device group.
NO ROOM, SKIPPING (file name)	No space is available on the output device to perform the transfer. Predeletion may already have occurred.
SYSTEM ERROR-CLOSING FILE	Directory format or system has become corrupted.

3.6 CREATE COMMAND

The CREATE command calls and runs the OS/78 Editor to allow a new file to be opened. Once opened, the new file is ready to receive any text typed in. Only one output file specification is allowed.

Format:

.CREATE outdev:file.ex/options

After pressing the RETURN key, the number sign (#) is displayed on the terminal, indicating that the Editor is ready to receive Editor commands. The CREATE command also allows the Editor options. (See Chapter 4 for a detailed explanation of the OS/78 Editor).

Example:

```
.CREATE RXA1:RUN1.PA
#
```

In this example, the CREATE command opens a file named RUN1.PA on output device RXA1 .

Each time a CREATE command is executed, the accompanying arguments (device and file name) are remembered in a temporary area. These remembered arguments allow the EDIT command to be typed without any arguments, allowing further editing of a file previously created (on the same day). However, the EDIT command does not recall any files without an assigned date. Therefore, the DATE command should be used prior to the CREATE command.

This command causes the execution of both the CCL.SV and EDIT.SV programs.

3.7 CREF COMMAND

The CREF command assembles a PAL program and produces a cross-reference listing usually on LPT.

Format:

```
.CREF outdev:file.ex<indev:file.ex/options
```

Example:

```
.CREF PROGA
```

The cross-reference listing contains the source program with the assembled instructions followed by the cross-reference table which contains every user-defined symbol and literal. These listings serve as an aid in writing, debugging and maintaining assembly language programs. CREF is the same as using the PAL command with the /C option specified. Note that if no output device is specified, LPT is the default output device.

This command causes the execution of the PAL8.SV,CCL.SV, and CREF.SV programs.

Table 3-5 CREF Options

Option Code	Meaning
/P	Disable pass one listing output. The output is reenabled when $ is encountered. Thus the $ and symbol table are printed if the /P option is used.
/U	Disable pass one listing output and the symbol table.
/X	Do not process literals. For programs with too many symbols and literals for CREF, this option may create enough space for CREF to operate.
/E	Do not eliminate the intermediate file CREFLS.TM that is output from the assembly and used as input to produce the CREF listing.
/M	Cross-reference mammoth files in two major passes. Pass one processes the symbols from A through LGnnnn; pass two processes the symbols from LHnnnn through Z and literals. This permits significantly large files to be cross-referenced.

Table 3-6 CREF Error Messages

Error Messages	Meaning
SYM OVERFLOW	More than 896 (decimal) symbols and literals were encountered during a major pass. Try again using the /M option.
ENTER FAILED	Entering an output file was unsuccessful – possibly output was specified to a read only device.
OUT DEV FULL	The output device is full (directory devices only).
CLOSE FAILED	CLOSE on output file failed.
INPUT ERROR	A read from the input device failed.
2045 REFS	More than 2044 (decimal) references to one symbol were made.
HANDLER FAIL	This is a fatal error on output.

3.8 DATE COMMAND

The DATE command enters the date specified in the command line into the system.

Format:

.DATE dd-mmm-yy

where

dd is a 2-digit decimal number representing the day,

mmm is a 3-character alphabetic string representing the month which must be the first three letters of the month, and

yy is a 2-digit decimal number representing the last two digits of the year.

Example:

```
.DATE 23-MAY-77
```

If the DATE command is typed without any arguments after the date has already been entered by a previous DATE command, the current day of the week and date are displayed.

Example:

```
.DA 1-JUN-77
.DA
WEDNESDAY JUNE 1, 1977
```

If the current date is entered into the system via the DATE command after booting, the only directory entry dates that are valid are those for the current year and seven years preceding the current date. Any dates prior to this time will be printed incorrectly.

The DATA command is used to date directory entries when creating or updating, to date directory listings, and to date program output listings.

If the date is specified incorrectly, an error message is displayed.

Example:

```
.DA 23-05-77
BAD DATE
```

This command causes the execution of CCL.SV and is processed by the Monitor.

3.9 DEASSIGN COMMAND

The DEASSIGN command removes all user-defined names that were previously assigned to one or more of the permanent devices.

Format:

```
.DEASSIGN
```

Example:

```
.AS SYS DEV1
.DE
```

In this example, the system device can no longer be referenced as DEV1. Only SYS, DSK or RXA0 are valid.

The Monitor performs the DEASSIGN function.

3.10 DELETE COMMAND

The DELETE command deletes all specified files.

Format:

.DELETE indev:file1.ex...file5.ex/options

In response to the dot produced by the Monitor, type the files that are to be deleted in command line and press the RETURN key. This will delete the files from the specified device. Wildcards are permitted. Only one device may be specified.

Example:

```
.DEL RXA1:REPORT.FT,SECUR.FT,ASSIST

FILES DELETED:
SECUR.FT
REPORT.FT
ASSIST
```

A message followed by the name of each file deleted is displayed on the terminal.

NOTE

If no filenames are specified, *.* (meaning all files) is the default input specification.

This command deletes all copies of your system FORTRAN files which are on RXA1.

Example:

```
.DEL RXA1:*.*<SYS:*.FT
```

An advanced format that can be used with the DELETE command is

.DELETE outdev:file.ex<indev:file 1,.../options

If an output specification is given, then this command works differently. Instead of deleting the input files specified, a list is made up of those file names which match the given file specification, and that are on the input device. This list is then transformed to an output list by applying the wildcards specified in the output specification. The resulting list is then used to delete files from the output device specified.

This command causes the execution of both the CCL.SV and FOTP.SV programs.

Table 3-7 DELETE Options

/C	Delete only those specified files having the current date.
/O	Delete only these specified files with dates other than the current date.
/V	Delete all files that are not specified in the command line.
/Q	Each time a filename followed by a question mark is printed on the terminal, type a Y (yes) to delete the file or any other character (no) if the file is not to be deleted.
/N	Displays a log of all files matched on the output device to be displayed, but no files are actually deleted.

Table 3-8 DELETE Error Messages

BAD OUTPUT DEVICE	Self-explanatory. This message usually appears when a non-file structured device is specified as the output device.
BAD OUTPUT DIRECTORY	The directory on the specified output device is not a valid OS/78 device directory.
DELETES PERFORMED ONLY ON INPUT DEVICE GROUP 1 CAN'T HANDLE MULTIPLE DEVICE DELETES	More than one input device was specified with the /D option when no output specification (device or file name) was included.
ERROR READING OUTPUT DIRECTORY	Self-explanatory.
ERROR WRITING OUTPUT DIRECTORY	Self-explanatory.
ILLEGAL *	An * was entered as an embeded character in a file name, for example, TMP*.BN.
ILLEGAL ?	A ? was entered in an output specification.
NO FILES OF THE FORM xxxx	No files of the form (xxxx) specified were found on the current input device group.

3.11 DIRECT COMMAND

The DIRECT command sends the directory listings for file-structured devices to an output device. The terminal is the default output device.

Format:

```
.DIRECT dev:listfile<dev:filetype,.../options
```

Example:

```
.DIRECT RXA1:
```

Along with the necessary arguments for the command line are DIRECT options that can limit or expand the result of the DIRECT operation. The standard directory listing has the following format:

filename.ex	**nnn**	**dd-mmm-yy**
(filename with extension)	(number of blocks used in decimal)	(day, month, year)

NOTE

If the date is not entered into the system via the DATE command, directory listings will not contain file creation dates.

If no output device is specified, the output defaults to the terminal. However, if you want the directory printed on LPT, use the CCL option -L in the command line.

Example:

```
.DIR RXA1:-L
```

is equivalent to

```
.DIR LPT:<RXA1:
```

Wild cards are permitted in the input specification but are not allowed in the output specification.

Example:

```
.DIR SYS:*.FT
```

As a result, only those files with FT extensions in the directory are displayed on the terminal. For example,

```
.DIRECT RXA1:WN????.*
```

will display all the files from the directory on RXA1 that have a file name two to six characters long beginning with WN, and having any extension.

To save the directory listing and store it in a file, specify the file name and device that it is to be stored on.

Example:

```
.DIR RXA1:DIRECT.DI<SYS:
```

3.11.1 Considering the Date in a Directory Listing

Use the /C option to display the names of the files that have the current date on the terminal. To display only those file names from the directory with a date other than the current date, use the /O option in the command line.

3.11.2 Choosing the Directory Format

To display just the file names from the directory, use the /F option in the command line. The /F option displays only the file names, and omits all block lengths and dates.

The /E option displays file names with their extensions, lengths, and dates, and empty files with their lengths in the exact order they are on the diskette.

When the /M option is specified, each empty file and its file length is displayed. Do not mix the /M option with the other options.

The =n option displays the list of file names in columns. The number of columns is determined by the decimal number specified as n. The values 1 through 7 can be substituted for n. For use on the terminal, values 1 through 3 are suggested. Values more than 3 are normally used for the line printer. Two columns is the default value. This value can be changed via the SET command. See the SET command for additional information.

To display a specific file name and all file names from that point on, use the /R option in the command line. This option should not be mixed with other options.

If there is a large number of file names from the directory to be displayed, use the /V option. By specifying the /V option, all file names that are not specified by the command line are displayed.

The starting block numbers (octal) of each file in the directory are displayed by using the /B option. The starting block numbers are in the form of nnnn and displayed following the file name and extension.

This command causes the execution of both the CCL.SV and DIRECT.SV programs.

Table 3-9 DIRECT Options

/B	Include the starting block numbers (octal) for each file in the directory listings.
/C	List only files with the current date, that is, the date entered with the most recent DATE command.
/E	Include empty file spaces in the directory listing.
/F	List only the file names and omit the lengths and dates.
/M	List only the empty files from the directory.
=n	List the directory in the number of columns specified by n. The n is a number ranging from 1-7.
/O	List only those file names from the directory with dates other than the current date.
/R	List the file name specified in the command line and the names of any files stored on disk following that specified file name.
/U	List each input specified as a separate directory listing with the file lengths and dates in the exact order specified in the command line.
/V	List all the files on the input device that are not specified in the command line.
/W	Display the current version number of DIRECT.

Table 3-10 DIRECT Error Messages

Message	Meaning
BAD INPUT DIRECTORY	This message occurs when the directory is non-existent or not in legal format.
DEVICE DOES NOT HAVE A DIRECTORY	This input device is a non-directroy device, for example, TTY or LPT. DIRECT can only read directories from file-structured devices.
EQUALS OPTION BAD	The =n option is not in the range 0-7.
ERROR CLOSING FILE	System error.
ERROR READING INPUT DIRECTORY	A disk-read error occurred while trying to read the directory.
ERROR WRITING FILE	A disk-write error occurred while trying to write the output file.
ILLEGAL *	An asterisk (*) was included in the output file specification or an illegal * was included in the input file name.
ILLEGAL ?	A question mark (?) was included in the output file specification.
NO ROOM FOR OUTPUT FILE	The output device does not have sufficient space for the directory listing file to be written.

3.12 DUPLICATE COMMAND

The DUPLICATE command copies or transfers the entire contents of one diskette to another diskette.

Format:

.DUPLICATE outdev:<indev:/options

In addition to the necessary arguments in the command line, DUPLICATE options can be used to affect the DUPLICATE operation.

Example:

```
.DUPLICATE RXA1:<RXA0:
```

The contents of input device RXA0 is copied onto output device RXA1.

3.12.1 Changing Devices Before and After Executing the DUPLICATE Command

You can only duplicate from RXA0 to RXA1 or RXA1 to RXA0. Driver RXA2 and RXA3 is not supported by the DUPLICATE command. Since the Monitor resides on the system device, the system device must remain on line when interacting with the Monitor and any OS/78 system programs.

If you want to transfer the contents of a diskette containing only files (one that does not contain a system), use the /P option. The /P option pauses before and after its execution of the DUPLICATE command. A ready message followed by a question mark is displayed. This pause provides time to remove the system device and mount a device onto which the contents are to be transferred, to or from. To start the DUPLICATE operation, type a Y after the question mark and press the RETURN key. After the DUPLICATE operation is completed, a message is displayed asking if the Monitor is remounted. The second pause provides time to remove the new device and remount the system device so control can return to the Monitor. After remounting the system device, type a Y after the question mark and press the RETURN key, to return control to the Monitor.

Example:

```
.DU RXA0:<RXA1:/P
READY?Y
IS MONITOR REMOUNTED?Y
.
```

3.12.2 Performing a Read Check

To check the integrity of a diskette, use the /R option and specify only the input device. By specifying the /R option, every block of the specified device is read and checked for bad sectors. If bad sectors exist, a message with the device, track number and sector number is displayed. If none exist, control returns to the Monitor.

Example:

```
.DU <RXA1:/R
INPUT DEV READ ERROR TRACK:nn SEC:nn
```

3.12.3 Transfer Without Checking for Identical Contents

To transfer the contents of one device to another without performing any checks, use the /N option. By specifying the /N option, the contents of the input device is transferred or copied to the output device. If /N is not specified and the DUPLICATE operation is completed, the contents of the output device is compared to that of the input device to assure accuracy.

3.12.4 Check for Identical Contents Without Transferring

To check the contents of devices to see if they are identical without transferring, use the /M option. By specifying the /M option the contents of both devices are read and checked for a match. If they do match, control returns to the Monitor. However, if they differ in any way, a message, the device name, the track number, and the sector number of the sectors or blocks that do not match are displayed.

Example:

```
.DU RXA1:<RXA0:/M
COMPARE ERROR TRACK nn SECTOR nn
```

This command causes the execution of both the CCL.SV and the RXCOPY.SV programs.

Table 3-11 DUPLICATE Options

Option	Meaning
/P	Pause and wait for user response before and after data transfers to/from diskette.
/N	Copy the contents of one device to another but do not check them for identical contents unless otherwise specified.
/M	Check both devices for identical contents and list the areas that do not match but do not perform a transfer unless otherwise specified.
/R	Read every block on the input device and list the bad sectors but do not perform a transfer unless otherwise specified.
/V	Display the current version number of the program invoked by the DUPLICATE command.

Table 3-12 DUPLICATE Error Messages

Message	Meaning
NO INPUT DEVICE	No input device is specified.
CAN'T LOAD INPUT DEVICE	The name of the input device specified in the command line is not a permanent device name.
CAN'T LOAD OUTPUT DEVICE	The name of the output device specified in the command line is not a permanent device name.
COMPARE ERROR	When using the /M option all the areas that do not match are printed as COMPARE ERRORS.
INPUT DEVICE READ ERROR	Bad data on input device; bad sectors.
OUTPUT DEVICE READ ERROR	Bad data on output device; bad sectors.
OUTPUT DEVICE WRITE ERROR	Hardware disk-write error.
DEVICE IS NOT RX	Either the input or output device specified is not one of the following: RXA0: SYS: RXA1: DSK:

3.13 EDIT COMMAND

The EDIT command calls the OS/78 Editor which allows the editing of already existing files. These files must contain ASCII source code.

Format:

.EDIT outdev:file.ex<indev:file.ex/options-ex

In response to the dot produced by the Monitor, type the necessary command line and press the RETURN key. The file is now open and the output file is ready for modification. After pressing the RETURN key, the number sign (#) is displayed on the terminal, informing you to proceed with an Editor input command which is almost always a read (#R) command. (See Chapter 4 for a detailed explanation of the OS/78 Editor).

Example:

```
.EDIT RXA1:FILE1.FT<RXA0:TABLE.FT
#
```

In this example, the EDIT command opens output file FILE1.FT an output device RXA1: and opens input file TABLE.FT on input device RXA0:.

NOTE

Since no output device is specified, the default option (DSK:) is used as the output device.

The EDIT command can recall arguments from a previous CREATE or EDIT command. This allows the EDIT command to be used without any arguments. However, the command must be used on the same day the file was created because the command does not remember arguments that were typed on a previous day. However, if you specify both an input and an output file, only the arguments up to but not including the left-angle bracket (<) are remembered.

Example 1:

```
.DA 14-JUN-77
.DA
TUESDAY JUNE 17, 1977
.CREATE MATH.PA
#A
   .
   .
   .
#E
.EDIT
#R
```

(MATH.PA is remembered by the EDIT command)

```
   .
   .
   .
#E
```

The above example shows a sequence of commands in creating and editing of file MATH.PA. The EDIT command is given without any argument since it remembers the file MATH.PA specified with the CREATE command.

If the date has changed and a command line is typed containing the EDIT command without any arguments, the error message BAD RECOLLECTION will be displayed.

Example 2:

```
.EDIT A2.FT<A1.FT
#R
 .
 .
 .
#E
 .
 .
 .
.EDIT
```

Since the second EDIT command had no arguments, the previous argument up to the left-angle bracket (<) is recalled. Thus, the second EDIT command is equivalent to:

```
.EDIT A2.FT
```

This command causes the execution of both the CCL.SV and EDIT.SV programs.

3.14 EXECUTE COMMAND

The EXECUTE command performs the following functions:

1. assemble/compile, link, load and execute a source program;
2. link, load and execute an already assembled or compiled program; or
3. execute an already linked and loaded program.

Format:

.EXECUTE outdev:file.ex<indev:file.ex/options-ex

To use the EXECUTE command with a source file, the input device must be specified. Then that source file is assembled or compiled, linked and loaded into memory and executed. The file extension used determines the compiler or assembler called.

If the file name extension is not a standard extension, use the correct assembler or compiler extension as a CCL option in the -ex portion of the command line as follows:

Example:

```
.EXE RXA1:NABS.IN-FA
```

If the EXECUTE command is used with an already assembled or compiled program, the input device must be specified. When the EXECUTE command is processed, the binary file is loaded into memory and executed. If the file name extension is not specified in the command line but is specified in the directory, the correct extension is automatically added.

The EXECUTE command can remember arguments from a previous COMPILE, LOAD, PAL, or EXECUTE command. For the EXECUTE command to recall remembered arguments, the EXECUTE command must be used on the same day the remembered arguments were stored in the temporary file. The reason for this is that the command does not remember arguments typed on a previous day.

NOTE

For options acceptable to the EXECUTE command, refer to the appropriate language section.

This command causes the execution of CCL.SV and one of the following: PAL8.SV and ABSLDR.SV, F4.SV and LOAD.SV, or BCOMP.SV and BLOAD.SV.

3.15 GET COMMAND

The GET command loads memory image files with .SV extensions from an input device back into the same location in memory from which it was saved.

Format:

```
.GET indev:file.ex
```

Example:

```
.GET RXA0:JOBCNT
```

If you specify a file name with no extension, an .SV extension is assumed.

Example:

```
.GET RXA1:FILE3
```

In this example, FILE3.SV is loaded into memory.

The specified input file along with its Job Status Word is loaded into memory. The entire Core Control Block is sent to the system device where it can be referenced and maintained (see Section 2.8.4).

The GET command is commonly used to load save files from a device into memory. These save files can be executed either by the START or EXECUTE command.

In addition, they can be debugged by the OS/78 octal debugging program (see ODT command).

NOTE

The colon used with the permanent device name can be replaced with a space.

The Monitor performs the GET function.

3.16 HELP COMMAND

The HELP command accesses a specified HELP file and displays its contents on the terminal. These HELP files contain useful information about the OS/78 program you specified in the command line (see Table 3-14).

Format:

.HELP outdev:file.ex<argument

where

argument is usually an OS/78 command.

NOTE

If no output device is specified, the terminal (TTY) is assumed.

For the current HELP files available, see the list containing all OS/78 HELP files below.

If no output file extension is specified the .HL extension is assumed. There is a HELP file which contains all OS/78 commands. This file is displayed by typing the HELP command without any arguments as follows:

```
.HELP
```

If a HELP file for a specific command is desired, type the name of the command for which the information is desired. For example,

```
.HELP PAL
```

To obtain a list of all legal arguments for HELP, type the HELP command followed by an asterisk or type the HELP command followed by 'HELP' as follows:

```
.HELP *
```

or

```
.HELP HELP
```

Table 3-13 HELP Error Messages

Message	Meaning
NO HELP	There is no HELP.HL file on the system device.
NO TTY HAND	There is no handler in the system for the terminal.
NO HELP FILE	The input file specified does not exist as a HELP file.
READ ERR	An error occurred while reading the specified HELP file.
WRITE ERR	An error occurred while writing to the output file.
FETCH ERR	Cannot load handler for the specified device.
DEVICE FULL	The output device becomes full when writing or is already full before output begins.

Table 3-14 Available OS/78 Help Files

ABSLDR	LIST
ASSIGN	LOAD
BASIC	MAP
COMPARE	ODT
COPY	PAL
CREATE	R
CREF	RENAME
DATE	RUN
DEASSIGN	SAVE
DELETE	SQUISH
DIRECT	START
DUPLICATE	SUBMIT
EDIT	TERM
F4	TYPE
GET	ZERO

3.17 LIST COMMAND

The LIST command prints the contents of the specified input files on LPT.

Format:

.LIST indev;file1.ex...file5.ex/options

Example:

```
.LI PROD.BA
```

Along with the necessary arguments for the command line are options that affect the LIST operation.

When you use the LIST command, the contents of each input file is listed on LPT in the same order the file names are specified in the command line.

Example:

```
.LI RXA1:FL1.FT,FL2.FT,MASFL.FT
```

Although the format specifies that only five files can be used in the command line, more than five files can be listed on the line printer through the use of wild cards.

3.17.1 Using LIST Options

You can put a restriction on the files you want by specifying the LIST options in the command line. By specifying the /C option, all the specified input files with the current date are listed. If the /O option is used, all specified input files with dates other than the current date are listed. If the /V option is used all files other than those you specified in the command line are listed. If you do not know which files you want to go through the directory, file by file, use the /Q option. By specifying the /Q option, each file name followed by a question mark is printed on the terminal. Then you must decide whether or not that file is a desired file for the LIST operation. If yes, type a Y and that file is listed. If no, type any other character and that file is ignored. After your file is either listed or ignored, the next file name followed by a question mark is printed on the terminal. This questioning continues until all specified input files have been processed.

Example:

```
.LIST*.PA/Q
TOT.PA?Y
MASFIL.PA?N
UPDT06.PA?N
FINTOT.PA?Y
```

As a result, the files named MASFIL.PA and UPDT06.PA are not listed.

This command causes the execution of both the CCL.SV and FOTP.SV programs.

Table 3-15 LIST Options

/C	List only those files having the current date.
/O	List only those files having a date other than the current date.
/V	List all the files not specified in the command line.
/Q	Each time a file name followed by a question mark is printed on the terminal, type a Y (yes) to list the files or any other character (No) if the file is not to be listed.

Table 3-16 LIST Error Messages

BAD INPUT DIRECTORY	The directory on the specified input device is not valid OS/78 device directory.
BAD OUTPUT DEVICE	Self-explanatory. This message usually appears when a non-file structured device is specified as the output device.
ERROR ON INPUT DEVICE, SKIPPING (file name)	The file specified is not transferred, but any previous or subsequent files are transferred and indicated in the new directory.
ERROR ON OUTPUT DEVICE, SKIPPING (file name)	The file specified is not transferred, but any previous or subsequent files are transferred and indicated in the new directory.
ERROR READING INPUT DIRECTORY	Self-explanatory.
ILLEGAL *	An * was entered as an embeded character in a file name, for example, TMP*.BN.
ILLEGAL ?	A ? was entered in an output specification.
NO FILES OF THE FORM xxxx	No files of the form (xxxx) specified were found on the current input device group.

3.18 LOAD COMMAND

The LOAD command loads either an absolute binary file with a .BN extension or a relocatable FORTRAN binary file with an .RL extension into memory.

3.18.1 Absolute Binary Files

Absolute binary files are loaded into memory and are ready for execution.

Once a program file is loaded into memory, the following three things can be done.

1. Start the program, using the START command.
2. Save the program, using the SAVE command.
3. Examine or debug the program, using the ODT command.

Format:

.LOAD indev:file1.ex...file9.ex/options

Along with the necessary arguments for the command line are options for absolute binary file that can affect the LOAD operation.

Example:

```
.LOAD FL3.BN,RUNSEQ.BN
```

NOTE

Use the ODT command to examine memory location after the LOAD command is entered and your files are loaded.

If all concatenated programs with .BN extensions located in a single file are to be loaded into memory, use the /S option in the command line. By specifying the /S option, not only the first program found with a .BN extension in a single file but all programs with .BN extensions are loaded into memory.

3.18.2 Loading Programs in Specified Areas

Absolute binary files can be loaded into a field other than the one they would normally load into by using /n option, where n is an octal number 0 through 3 which represents a field. By specifying the /n option, all specified input files are loaded beginning with the indicated field at a loading address equal to the address specified in the file.

Both the field and the starting address can be specified by using the =fnnnn option

where

f — is an octal number 0 through 3 which represents the field, and

nnnn — is a 4-digit octal number 0000 through 7777 which represents a starting address.

NOTE

If fnnnn is all zeros, the default option (field 0 starting address 0200) is used.

If locations 0-1777 in field zero are not being used by the program, use the /8 option in the command line. By specifying the /8 option, the contents in these locations are not saved nor is the Command Decoder (system program) reloaded. In effect, these unnecessary operations are not performed.

If locations 0-1777 in field one are not being used by the program, use the /9 option in the command line. By specifying the /9 option, the contents in these locations are not saved nor is the User Service Routine (system program) reloaded. In effect, these unnecessary operations are not performed.

3.18.3 Editing or Patching a SAVE File

Changes can be made to the contents of files after they are already assembled and saved on a device by using the /I option in the command line. By specifying the /I option, SAVE files are loaded and treated as absolute binary files. This allows patches to be made to an already saved file without reassembling the entire program.

3.18.4 Execution After Loading

If execution of the program is to follow immediately after the loading of the program use the /G option in the command line. By specifying the /G option, execution takes place immediately after loading is completed. Control is transferred to the executing program and remains there until completion or until a transfer to the Monitor from within the program takes place.

3.18.5 Clearing Memory After Loading

If you have already loaded a specific number of files and want to change either some or all of the files loaded into memory, use the /R option in the command line. By specifying the /R option, the portion of memory where your files were loaded is reset or cleared to appear as though nothing had been loaded.

When absolute binary files are loaded using the LOAD command, both the CCL.SV and ABSLDR.SV programs are executed.

3.18.6 Relocatable FORTRAN Binary Files

Since relocatable FORTRAN binary files are dependent programs, both files and system programs or subroutines are loaded and linked in memory.

Format:

```
.LOAD outdev:file.ex<indev;file.ex/options-ex
```

After files are loaded, the system library FORLIB.RL is searched for programs or subroutines that are referenced by the program. When the subroutines are found, they are loaded and linked to the program.

If you have your own library file and have referenced several subroutines from that file, use the /L option in the command line. By specifying the /L option, the first input file in the command line is used as a library file and any remaining input files in the command line are ignored. The directory is searched for the files which, once found, are loaded in memory and linked to your program.

NOTE

A FORTRAN program can be compiled, linked, and executed by using the COMPILE command with the /L and /G options. This eliminates the need for using the LOAD and EXECUTE commands.

If you have more than nine FORTRAN input files to load, use the /C (continue) option in the command line. By specifying the /C option, the remaining files are accepted as valid input and should be continued on the next entry line.

If a symbol map output file is specified in the command line with a .RL file and you want the system symbols also included in the map, use the /S option in the command line. By specifying the /S option, all system symbols are included in the symbol map and marked by a preceding number sign (#).

When relocatable binary files are loaded and linked using the LOAD command, both the CCL.SV and LOAD.SV programs are executed.

3.19 MAP COMMAND

The MAP command produces a map of all the memory locations used by the specified absolute binary files. The LPT line printer is the default output device for this command. Terminal output may be requested through the use of the CCL option -T.

Format:

.MAP outdev:file.ex<indev:file1.ex. . .file9.ex/options-ex

In addition to the necessary arguments are MAP options that can affect the MAP operation.

NOTE

If a filename without an extension is specified as the output file, an .MP extension is automatically appended. Input extensions default as described for the LOAD command.

3.19.1 MAP Output

The output of the MAP operation is a series of lines, each of which is comprised of a string of digits. Each digit represents a single memory location, and can have the value 0, 1, 2 or 3. The value means the following:

0 means that the location was not loaded into.
1 means that the location was loaded into once.
2 means that the location was loaded into twice.
3 means that the location was loaded into three or more times.

Appearance of a 2 or 3 may imply a programming error (for example, two separate routines are each trying to load values into the same location).

Each line of digits represents 64 (decimal) or 100 (octal) memory locations. All lines are blocked in groups of two, whereby, each group represents a page. If the resulting map is sent to the terminal, it is bordered on top by a set of octal digits which range from 00-77. Each pair of octal digits corresponds to one memory location (see following example).

It is also bordered down the left hand column by a set of octal digits which are multiples of 64(decimal). Each set of octal digits corresponds to the first memory location on that line. To determine the address (relative to zero) of a specific location in memory, find the line containing that location. Take the set of octal digits bordering that line and the pair of octal digits that specific memory location falls under and add both values together.

The following is an example of output on a terminal:

```
BITMAP V4 FIELD 0

      0000000011111111222222223333333344444444555555556666666677777777
      0123456701234567012345670123456701234567012345670123456701234567

00000 2222222221111111100000000000000000000000000000000000000000000000
00100

00200 2222222222222222222222222222222222222222222222222222222222222222
00300 2222222222222222222111111111111111111111111111111111111112222222

00400 2222222222222222222222222222222222222222222222222222222222222222
00500 2222222222222222222222222222222222222222222222222222222222222222
```

```
00600 1111111111111111111111111111111111111111111111111111111111111111
00700 1111111111111111111111111111111111111111111111111111111111111111
01000
01000
01100

01200
01300

01400
01500

01600
01700
.
.
.
```

In this example, the location resulting from the addition is 322.

The following is an example of output directed to a line printer:

```
BITMAP V4 FIELD 0

00000 22222222 21111111 10000000 00000000 00000000 00000000 00000000 00000000
00100

00200 22222222 22222222 22222222 22222222 22222222 22222222 22222222 22222222
00300 22222222 22222222 22111111 11111111 11111111 11111111 11111111 11122221

00400 22222222 22222222 22222222 22222222 22222222 22222222 22222222 22222222
00500 22222222 22222222 22222222 22222222 22222222 22222222 22222222 22222222

00600 11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111
00700 11111111 11111111 11111111 11111111 11111111 11111111 11101111 11111111

01000
01100

01200
01300

01400
01500

01600
01700
.
.
.
```

If any of the specified input files contain more than one program, only the first binary program in each file is used in the MAP command. However, by using the /S option in the command line, all binary programs in the specified input files are used in the MAP command. If there is more than one program being mapped, and you want all maps stored in one memory field, use the /n option, where n is an integer representing a field number. By specifying the /n option, all the maps of programs in the specified input file are constructed in the field you specified as n.

If more than nine input files are to be specified in the MAP command, specify only nine input files, then press the ESCape key. This causes the Command Decoder to print an asterisk (*) which allows you to continue to specify more input files, each terminated by the RETURN key. If it is realized at this point that a wrong input file was specified, use the /R option at the end of that line. By specifying the /R option, memory is reset to look as though nothing has been read in. This is shown in the following example; the second ESCape causes the command to be executed.

Example:

```
.MAP A,B,C,D,E,F,G,H,I (ESC)
*J
*K
*L/R (ESC)
```

If only a terminal is available and no line printer, the style of output on the terminal can be changed to look as though it came from a line printer by using the /T option. Similarly, the style of output on the line printer can be changed to appear as though it came from the terminal.

This command causes the execution of both the CCL.SV and BITMAP.SV programs.

Table 3-17 MAP Options

Option Code	Meaning
/R	Reset the map just constructed in memory to look as though nothing has been read in.
/S	Instead of mapping only the first binary file contained in the specified input files, map every absolute binary file in the specified input files.
/n	Confine the construction of all maps to the field you specify as n.
/T	If output is sent to the terminal, change the format of the map to that of the line printer and if output is sent to the line printer, change the format of the map to that of the terminal.

Table 3-18 MAP Error Messages

Message	Meaning
I/O ERROR FILE #n	An I/O error occurred in input file number n.
BAD INPUT, FILE #n	A physical end of file has been reached before a logical end of file, or extraneous characters have been found in binary file n.
BAD CHECKSUM, FILE #n	File number n of the input file list had a checksum error.
NO INPUT	No binary file was found on the designated device.
ERROR ON OUTPUT DEVICE	Error occurred while writing on output device.
NO /I	Cannot produce a map of an image file.

3.20 MEMORY COMMAND

The MEMORY command is used to find out the highest memory field available to the OS/78 system in hardware or to limit that value in software.

Format:

.MEMORY n or .MEMORY

where

n is an octal number representing the number of fields (4K) available to be used by OS/78. It is in the range of 0-3.

Example:

```
.MEMORY 3
16K MEMORY
```

The following table lists the values of n and their meanings.

n	memory
0	all available memory (16K)
1	8K
2	12K
3	16K

To find the amount of memory actually being used by OS/78, type the command with no argument.

```
.MEM
12K/16K MEMORY
```

In this example, a 16K system has been restricted to only 12K of available memory. This was done by using a MEMORY 2 command.

If all available memory is being used, the total amount of memory is printed.

Example:

```
.MEM
16K MEMORY
```

This command causes the execution of the CCL.SV program.

3.21 ODT COMMAND

The ODT command allows you to debug a program currently in memory, control its execution, and make alterations to the program by typing ODT instructions at the terminal.

Format:

```
.ODT
```

The current program in memory is available for examination and modification of locations in memory and controlled execution through the use of ODT instruction breakpoints. For more detailed information on ODT, see Chapter 9.

The Monitor processes the ODT command.

Example:

```
.ODT
```

3.22 PAL COMMAND

The PAL command assembles a PAL8 source file, producing an absolute binary file with a .BN extension.

Format:

.PAL outdev:file.ex<indev:file.ex/options-ex

or

.PAL bin,list<input specification

to output a binary and a listing file

or

.PAL bin,list,cref<input specification

to output a binary, listing, and CREF file.

Example:

```
.PAL ASSEM1.PA
```

If a file name is specified without an extension in the command line, .PA is assumed.

The PAL command can recall arguments from a previous COMPILE, LOAD, or EXECUTE command. (See Chapter 5 for information on the PAL8 assembler.)

This command causes the execution of both the PAL8.SV and CCL.SV programs.

3.23 R COMMAND

The R command loads the specified SAVE file in memory at its starting address and initiates execution.

NOTE

The difference between the R and RUN command is that the R command assumes the program is on device SYS, while the RUN command makes no such assumption. In addition, the RUN command writes on SYS during execution to save the program's Core Control Block for subsequent use in saving and/or starting the programs. The Core Control Block is not transferred to the system device when the R command is executed.

Format:

```
.R file.ex
```

Note that a device name does not have to be given.

Example:

```
.R TEST
```

When a file name is specified without an extension, a .SV extension is assumed.

The R command is most commonly used on programs that are not going to be resaved. It is suggested that the RUN or GET commands be used with programs that will eventually be updated (using, for example, ODT) and saved.

The Monitor processes the R command.

3.24 RENAME COMMAND

The RENAME command changes the name of the input file to the name specified as the output file. The same device must be specified in both the input and output specifications.

Format:

```
.RENAME dev:file.ex<dev:file.ex
```

Execution of the RENAME command changes the file name, and a message followed by the old file name is displayed on the terminal.

Example:

```
.REN DSK:FILE.FA<DSK:RECORD.PA

FILES RENAMED:
RECORD.PA
```

This new file name replaces the old file name in the input directory. The creation date and the contents of the input file remain the same.

Wildcards may be used with this command.

Example:

```
.RENAME *.BK<*.FT
```

renames all files with .FT extensions to .BK extensions.

Table 3-19 RENAME Options

Option Code	Meaning
/C	Rename the input file only if it has the current date.
/O	Rename the input file only if it has a date other than the current date.
/V	Rename all files other than the one specified in the command line.
/T	After renaming the file, change the date of the new output file to today's date.

3.25 RUN COMMAND

The RUN command loads the specified memory image (.SV) file in memory, transfers its Core Control Block onto the system device, and then initiates execution at the program's starting address.

Format:

```
.RUN indev:file.ex
```

Example:

```
.RUN RXA1:PROG.
```

If a file name is specified without an extension, the extension ".SV" is assumed.

NOTE

The RUN command is equivalent to a GET command followed by a START command.

See the R command for further information.

NOTE

The colon used with the permanent device name can be replaced with a space.

The Monitor processes the RUN command.

3.26 SAVE COMMAND

The SAVE command saves the portion of memory being used. The memory used is indicated by the contents of the Core Control Block saved on SYS in a Monitor area or as explicitly specified by you. This new memory image file is given a default .SV extension if an extension is not specified.

Format:

```
.SAVE input dev:filename.ex fnnnn-fmmmm,fpppp,. . .;fssss=cccc
```

where

fnnnn-fmmmm — fnnnn is a 5-digit octal number representing the beginning address of a contiguous portion of memory to be saved. The beginning address consists of a field number (f) followed by the memory location within the field,

fmmmm is a 5-digit octal number representing the ending address of the portion in memory. The ending address consists of the same field number as the beginning address (f) followed by the memory location within the field;

NOTE

The beginning and ending address must have the same field number. Crossing field boundaries is not permitted.

fpppp — fpppp is a 5-digit octal number representing the address of one location in memory. This represents saving the entire page in which the given location occurs;

;fssss — ;fssss is a 5-digit octal number representing the starting address of the program to be saved. The starting address consists of a field number (f) followed by the memory location within the field; and

=cccc — =cccc is a 4-digit octal number representing the contents of the Job Status Word.

The Job Status Word indicates what parts of the file use memory and how as shown in Table 3-20.

Table 3-20 Job Status Word

Bit Condition	Meaning
Bit 0=1	File does not load into locations 0-1777 in field 0, (0000-1777).
Bit 1=1	File does not load into locations 0-1777 in field 1, (10000-11777).
Bit 2=1	Program must be reloaded before it can be restarted because it modifies itself during execution.
Bit 3=1	Program being run will not destroy the BATCH monitor.
Bit 4=1	A memory image file that was generated through the LINKER contains overlays.
Bit 5	Reserved for OS/78 system programs
Bits 6-9	Unused, and reserved for future expansion.
Bit 10=1	Locations 0-1777 in field 0 need not be saved when calling the Command Decoder overlays.
Bit 11=1	Locations 0-1777 in field 1 need not be saved when calling the USR.

NOTE

If these arguments are not specified in the command line, the arguments are automatically taken from the current Core Control Block, and stored by the last GET, RUN, LOAD or EXECUTE command.

3.26.1 Restrictions on Arguments of the SAVE Command

There are several restrictions on the arguments used with the SAVE command. They are as follows:

1. The output device must be explicitly specified in the command line. It does not default to DSK.

```
.SAVE DSK:FILE1.PA
```

2. The beginning and ending addresses of an area in memory (fnnnn-fmmmm) must both be located in the same field. Crossing field boundaries is not permitted.

   Example:

```
.SAVE SYS EXAMPL.BN 20055-20643
```

3. Once an area on a page is specified, that entire page is saved. Therefore, if another area on the same page is specified, an error message is sent to the terminal informing you that page is already saved.

```
.SAVE RXA1:FL2 10077-10122
.SAVE RXA1:FL2 10156-10177
```

4. If a field number is not specified in the starting and ending addresses of an area in memory, field zero is used. Otherwise, the field number must be specified.

```
.SAVE SYS FILE1 0-177,201-377
```

5. Saving the area between locations 7600-7777 in any field is permitted. However, locations 7600-7777 in fields 0 and 1 should not be saved because the system resident Monitor code resides there. Therefore, if the area between 7600-7777 is going to be saved, limit it to fields two and three and even then only if necessary (due to BATCH).
6. If you specify an address on a page having an odd page number, that page cannot be saved without the previous page also being saved. This is automatically done by the system.

   Example:

```
.SAVE DSK PROG1 2434
```

   In this example, the location is on page 12 which is even. Thus, the contents of page 12 are saved.

   Example:

```
.SAVE DSK PROG2 2634
```

   In this example, the address 2634 is on page 13 which is odd. Therefore, in order to save page 13, page 12 is also saved. This means that the effective area being saved is 2400-2777.

NOTE

The colon used with the permanent device name can be replaced with a space.

The Monitor processes the SAVE command.

3.27 SET COMMAND

The SET command modifies the operating characteristics of OS/78, according to options that are specified.

Format:

.SET device [NO] attribute [argument]

where

device — indicates the handler of OS/78 that is to be modified, such as: SYS, TTY, or LPT;

[NO] — indicates that the feature specified is to be turned off; NO cannot be used with every attribute;

attribute — is the feature to be modified (See the SET command attributes); and

[argument] — is a variable (numeric or alphabetic) that is supplied by you.

Table 3-21 SET Command Attributes

TTY	SYS	LPT	Any Device
READONLY WIDTH = n LC PAGE ECHO SCOPE HEIGHT m PAUSE n COL n ARROW ESC	INIT xxxxx	WIDTH = n LC	READONLY

3.27.1 ECHO

When used with the NO modifier, this attribute removes TTY character echoing from the TTY handler. This does not affect monitor I/O.

Format and Example:

```
.SET TTY NO ECHO
```

To restore character echoing, use the ECHO attribute in the command line.

Example:

```
.SET TTY ECHO
```

3.27.2 ARROW

When used with the NO modifier, this attribute forces each control character to be printed on the terminal in the form:

↑X

where:

↑ represents the CTRL key

X represents the control letter

Format and Example:

```
.SET TTY NO ARROW
```

For a CTRL C character, a ↑C is printed on the terminal.

To remove this attribute, use the ARROW attribute in the command line.

Example:

```
.SET TTY ARROW
```

3.27.3 SCOPE

When an input line of characters is typed on the terminal and echoed on the screen, this attribute erases the last character echoed on the screen each time the DELETE key is pressed.

Format and Example:

```
.SET TTY SCOPE
```

To remove this attribute, use the NO modifier in the command line.

Example:

```
.SET TTY NO SCOPE
```

3.27.4 WIDTH = n

This command changes the width of the terminal to the argument you specify as n.

Format:

.WIDTH TTY WIDTH = n

where

n is a decimal number and a multiple of 8 which is in the range of 001-255

NOTE

The n must not be 128. Also, the NO modifier is not permitted to be used with this attribute.

Example:

```
.SET TTY WIDTH=72
```

This command can also be used with the LPT handler, in which case, LPT replaces TTY in the command string.

3.27.5 HEIGHT m

This attribute changes the number of lines that are printed on the terminal between pauses. Twenty four lines is the default value of m.

Format and Example:

```
.SET TTY HEIGHT 12
```

3.27.6 PAGE

When used with the NO modifier, this attribute removes the CTRL/S and CTRL/Q features (See Section 2.1.3).

Format and Example:

```
.SET TTY NO PAGE
```

To restore the CTRL/S and CTRL/Q features, use the PAGE attribute in the command line.

```
.SET TTY PAGE
```

3.27.7 ESC

This attribute restrains the printing of the dollar sign ($) on the terminal when the ESCape key is pressed. However, ESC does allow the terminal to receive its octal value (033_8) which puts the system into a different mode. Once it is in this mode, moving the cursor and erasing the terminal screen are some of the options that can be performed.

Format and Example:

```
.SET TTY ESC
```

To remove this attribute, use the NO modifier with the ESC attribute.

```
.SET TTY NO ESC
```

3.27.8 PAUSE

This attribute changes the pause time between terminal output frames from 3 seconds (its default) to the decimal number you specify as n.

Format and Example:

```
.SET TTY PAUSE 5
```

NOTE

If you specify zero as n, or use the NO modifier, no pause takes place.

3.27.9 READ ONLY

This attribute causes the device specified to become a read-only device.

Therefore, any output sent to this device will cause an error message informing you that the output device is a read-only device.

Format and Example:

```
.SET TTY READO
```

To remove the READONLY attribute, use the NO modifier in the command line.

```
.SET TTY NO READO
```

NOTE

The READONLY attribute remains in effect only until the next time you push the START button at which time its original status is restored.

3.27.10 COL n

This attribute changes the number of columns used to print the directory from two (its default) to the decimal number you specify as n.

Format and Example:

```
.SET TTY COL 3
```

To remove the value you specified as n and return to its default value of 2, you must specify 2 as the argument n.

```
.SET TTY COL 2
```

3.27.11 LC

This attribute causes the TTY or LPT handler to accept lower case characters on input.

Format and Example:

```
.SET TTY LC
```

To remove this attribute, use the NO modifier in the command line.

```
.SET TTY NO LC
```

3.27.12 INIT xxxxx

This attribute causes the system device to execute the command you specify as xxxxx when the START button is pushed. This command name can contain a maximum of five characters excluding a carriage return.

Format and Example:

```
.SET SYS INIT HELP
```

To execute the command contained in the file INIT.CM, type:

```
.SET SYS INIT
```

If you want the system to print the monitor dot immediately after the START button is pushed, type the following command:

```
.SET SYS NO INIT
```

Table 3-22 SET Error Messages

? SYNTAX ERROR Incorrect format used in SET command or NO specified when not allowed.
? UNKNOWN ATTRIBUTE FOR DEVICE dev An illegal attribute was specified for the given device.
? CAN'T – DEVICE IS RESIDENT No modifications are allowed to the system handler.
? CAN'T – OBSOLETE HANDLER The handler has an old version number.
? CAN'T – UNKNOWN VERSION OF THIS HANDLER The version of the handler is not one recognized, possibly because it is a newer version.
? ILLEGAL WIDTH A width of 0 or a width too large was specified; or for the TTY, a width of 128 or one not a multiple of 8,was specified.
? NUMBER TOO BIG The number specified was out of range.
? CAN'T – DEVICE DOESN'T EXIST A nonexistent device was referenced.
? I/O ERROR ON SYS: An I/O error occurred while trying to read or rewrite the handler.

3.28 START COMMAND

The START command initiates execution of the program that is currently in memory either at the specified starting address in the command line or the starting address specified in the current Core Control Block.

Format:

```
.START fnnnn
```

where

fnnnn is a 5-digit octal number representing the starting address of the program. The starting address consists of a field number (f) followed by the memory location (nnnn) which is within the field. If the starting address is not specified in the command line, the program is started at the address specified in the Core Control Block.

Example:

```
.START 10555
```

In this example, this command line will execute a program starting in field 1 at location 555.

The Monitor processes the START command.

3.29 SQUISH COMMAND

The SQUISH command eliminates any embedded empty files on the input device.

Format:

```
.SQUISH indev:
```

Whenever the SQUISH command is used, a message followed by a question mark that asks if this is the correct device for the SQUISH operation is displayed.

Example:

```
.SQ RXA1:
ARE YOU SURE?Y
```

If yes, type a Y and the files are compressed.

```
.SQ SYS:
ARE YOU SURE?N
```

If no, type any other character and the operation is ignored.

If the device is a system diskette, the system is preserved during the transfer. In order to eliminate the system, use the ZERO command (See ZERO command). If no output device is specified, then the specified device in the command line is itself squished.

Example:

```
.SQ SYS:
ARE YOU SURE?Y
```

In this example, all embedded empty files on the system device are eliminated.

NOTE

A diskette error during a SQUISH operation will leave the entire contents of the diskette corrupted in a non-obvious way. Therefore, this command should not be used unless a backup copy of both the system and other files is available.

This command causes the execution of both the CCL.SV and PIP.SV programs.

3.30 SUBMIT COMMAND

The SUBMIT command provides batch processing with the option of spooling to output files.

Format:

.SUBMIT spool dev:<DSK:batch file.ex/options

Example:

```
.SU RXA0:BTCHIN
```

If no input file extension is specified,.BI is assumed.

Along with the necessary arguments for the command line are SUBMIT options that can affect the result of the SUBMIT operation.

The SUBMIT command is commonly used to run multiple programs and sequences of system commands that require little or no interaction with the user or operator.

3.30.1 Processing and Terminating A Batch Input File

The SUBMIT command starts processing your batch input file by printing a job header. The job header is printed on the LA78 line printer if it is available. Note that the LQP78 line printer is not supported in batch processing. If an LA78 line printer is not available, the terminal is used as the output device to print a log of batch operations. Processing continues with the OS/78 commands and BATCH monitor commands following the $JOB batch monitor command and terminates with the $END batch monitor command. Termination can also occur from an error in the OS/78 commands. However, batch processing is allowed to continue in spite of errors if the /E option is used in the command line. By specifying the /E option, all errors are treated as non-fatal errors so that batch processing can continue uninterrupted.

If the ESCape key is used to terminate an OS/78 command in the batch file, it will cause your batch run to terminate unless the /E option is specified. To terminate all the commands in a batch file, use the RETURN key. If CTRL/C is typed during batch processing, the batch run is terminated and control returns to the Monitor.

If you want just the $JOB, and $MSG batch monitor commands, and all batch results sent to the output device, disregarding OS/78 commands and comment lines, use the /Q option in the command line. If you have a LA78 and want all batch output sent to the terminal instead, use the /T option in the command line. By specifying the /T option, batch output is sent to the terminal only.

If you want your batch input file to be processed without interruption, use the /U option in the command line. By specifying the /U option, any errors are non-fatal and your batch input file is processed without interruption. Anything you type on the terminal except CTRL/C during batch processing is ignored.

If you want your batch input file to be processed without echoing on the terminal and without sending the $JOB and $END batch monitor commands to the terminal and BATCH log, use the /H option in the command line. By specifying the /H option, echoing, $JOB, and $END batch monitor commands are suppressed. Statements that are not suppressed are as follows:

1. Any OS/78 command messages and user program messages sent to BATCH
2. User program messages sent to the terminal
3. Batch run-time error messages
4. $MSG statements

NOTE

Use either the /Q or /H option but not both in a command line.

3.30.2 Spooling

The SUBMIT command also provides optional spooling of output files. Spooling is a process whereby output to slow-speed devices (such as a line printer) is placed into queues on fast-speed devices (such as a diskette) until the slow speed devices are available for processing. One advantage of using spooling is a more efficient use of slow-speed devices, memory, and the central processor unit.

To use the spooling option during batch processing, specify an output device, the system disk as the input device and the batch file in the command line. The output device specified as the spool device must be a file-structured device (that is, diskette). Therefore, TTY and LPT should not be used. If no output device is specified, spooling does not take place.

Example:

```
.SU RXA1:<RXA0:BATCHX
```

NOTE

Your batch file must be on your system device (SYS).

When spooling is specified, all output to a non-file structured device is intercepted and stored in temporary files on the spool device. Every file that is spooled to your spool device is given a name. The first non-file structured output file is called BTCHA1. The second is called BTCHA2 and so on up to and including BTCHA9. If you have more than nine input files the names continue with BTCHB0 up to and including BTCHZ9.

See Chapter 8 for a detailed explanation of OS/78 Batch.

This command causes the execution of both the CCL.SV and BATCH.SV programs.

Table 3-23 SUBMIT Options

Option	Meaning
/E	Treat all errors as non-fatal errors so batch processing continues uninterrupted.
/Q	Send only the $JOB and $MSG batch monitor commands, and all results from batch processing to the output device.
/T	Send all output from batch processing to the terminal only.
/U	Do not interrupt batch processing and ignore anything typed at the keyboard with the exceptions of CTRL/C.
/H	Process the batch input file without echoing and without sending the $JOB and $END batch monitor commands to both the terminal and batch log.

3.31 TERMINATE COMMAND

The TERMINATE command causes the system to emulate a terminal without any knowledge of disk drives, processor or memory.

Example:

```
.TER
```

This command causes the execution of the CCL.SV program.

3.32 TYPE COMMAND

The TYPE command displays the contents of the specified input files on the terminal screen.

Format:

.TYPE indev:file1.ex...file5.ex/options

Along with the necessary arguments for the command line are TYPE options that affect the result of the TYPE operation.

The TYPE command displays the contents of each input file on the terminal in the same order the file names are specified in the command line.

Example:

```
.TY SYS:MASFL
```

Although the format specifies that only five file names can be used in the command line, more than five files can be typed on the terminal by using wildcards.

3.32.1 Using TYPE Options

Use the /C or /O options if the date is to determine what files are to be displayed. By specifying the /C option, all the specified input files with the current date are typed. If the /O option is used, all specified input files with dates other than the current date are typed. The /V option displays all files other than the ones that were specified. The /Q option allows each file name to be displayed followed by a question mark. If that file is the correct file for the TYPE operation, type a Y and that file is displayed on the terminal. If not, type any other character and that file is ignored. After the file is either typed or ignored, the next file name followed by a question mark is displayed on the terminal. This questioning continues until all specified input files have been typed.

NOTE

Use the LIST command to print files on the LA78 line printer (LPT).

This command causes the execution of both the CCL.SV and FOTP.SV programs.

Table 3-24 TYPE Options

/C	Type only those files with the current date.
/O	Type only those files with a date other than the current date.
/V	Type all files that were not specified in the command line.
/Q	Each time a file name followed by a question mark is printed on the terminal, type a Y (yes) to type the file or any other character (no) if the file is not to be typed.

Table 3-25 TYPE Error Messages

Message	Meaning
BAD INPUT DIRECTORY	The directory on the specified input device is not valid OS/78 device directory.
BAD OUTPUT DEVICE	Self-explanatory. This message usually appears when a non-file structured device is specified as the output device.
ERROR ON INPUT DEVICE, SKIPPING (file name)	The file specified is not transferred, but any previous or subsequent files are transferred and indicated in the new directory.
ERROR ON OUTPUT DEVICE, SKIPPING (file name)	The file specified is not transferred, but any previous or subsequent files are transferred and indicated in the new directory.
ERROR READING INPUT DIRECTORY	Self-explanatory.
ILLEGAL *	An * was entered as an embedded character in a file name, e.g., TMP*.BN.
ILLEGAL ?	A ? was entered in an output specification.
NO FILES OF THE FORM xxxx	No files of the form (xxxx) specified were found on the current input device group.

3.33 UA, UB, UC COMMANDS

The UA, UB, and UC commands are three separate commands that save their specified arguments in a temporary area. Only one argument is permitted with each command and this must be a single OS/78 command. These three commands have the capability to recall their arguments when the command that saved the OS/78 command is typed without any arguments. Once recalled, the OS/78 command is executed.

Format:

```
.UA )
.UB }   OS/78 command output
.UC )
```

In response to the dot produced by the Monitor, type one of the commands along with the OS/78 command you want saved.

Pressing the RETURN key saves the specified OS/78 command in a temporary area. There are 3 areas, one for each command. To recall and execute the stored OS/78 command, type the command that saved it, but do not specify any arguments.

Example:

```
.UA COPY RXA1:<DSK:RECALL
```

In this example, the OS/78 command, COPY and its arguments, is stored in a temporary file through the use of the UA command.

Example:

```
.UA
```

In this example the OS/78 command previously specified with the UA command and stored in a temporary file is recalled and executed.

These commands are most commonly used with OS/78 commands that are repeated throughout a batch file. The use of these instructions decreases the amount of input required. Note that only three OS/78 commands can be saved at one time.

The Monitor processes the UA, UB, and UC commands.

3.34 ZERO COMMANDS

The ZERO command clears the directory on a device, deleting any and all files stored on that device. Use this command with extreme caution, since it could destroy the contents of a system diskette. Use it only for diskettes that are used to store programs and data files. If used on a system diskette, it will destroy the system and convert it to a data diskette.

Format:

```
.ZERO dev:
```

This command causes the execution of both the CCL.SV and PIP.SV programs.

Example:

```
.ZERO RXA1:
```

By using the DIRECT command on the device just zeroed, a blank directory will appear showing the availability of 487 free blocks.

If the specified device is device SYS, a message followed by a question mark is printed on the terminal asking if this is the correct device for the ZERO operation. If yes, type a Y and the directory is zeroed. If no, type any other character and the ZERO operation is ignored. Remember that if a Y is typed, the system on the specified device will be destroyed.

NOTE

Never issue a ZERO SYS: command, as this will erase the system diskette.

CHAPTER 4
CREATING AND EDITING YOUR PROGRAM

4.1 INTRODUCTION

The Editor is a program that allows you to create and modify ASCII source files. These files may contain assembly language programs, FORTRAN and BASIC programs, or any other information that has the format of character strings.

The Editor is a very helpful tool; however, it must still be told precisely what to do. Its operation is directed by typing commands in the form of a single letter or a letter with arguments. Most commands are executed by pressing the RETURN key directly after the command line.

4.2 PAGE MAKEUP

The page of text is the fundamental unit into which the Editor considers a file to be divided. A page of text consists of all the ASCII characters between two form feed characters, not including the second form feed. In other words, a page begins after the form feed of the previous page, and ends with a form feed. A page is usually a physical page of a program listing.

A line of text consists of all the characters between two carriage return/line feeds, not including the second carriage return/line feed. All lines are implicitly numbered in decimal, starting with 1. This implicit enumeration is continually updated by the Editor to account for line insertions, moves, and deletions. For editing and listing purposes, each line is referred to by its current implicit decimal line number.

4.3 TEXT BUFFER

The text buffer is a storage area in memory that holds a text file one page at a time to allow the operations to be performed to create or modify the text. The editing commands will affect only the information that is placed into the text buffer.

The text buffer has room for approximately 5600 (decimal) characters. When text has been input so that there is room for only 256 (decimal) more characters in the buffer, the Editor causes the terminal buzzer to ring as a warning. From this point on, whenever a carriage return is detected during text input, the buzzer again sounds and the Editor prompts with the number sign (#), signifying it needs an editing command. Inputting one line at a time may continue until the absolute end of the buffer is encountered. At this point, no more text can be placed in the buffer, a question mark "?" is printed, and the Editor prompts with the number sign (#), indicating it has returned to the command mode. If text is segmented into pages as explained later in this chapter, there should always be sufficient room in the text buffer.

The Editor reads one page of text at a time from an input file into the text buffer where the page becomes available for editing. Then the Editor writes the text buffer onto the output file. Editing commands then perform the following functions:

1. Locate text to be changed (string and character searches).
2. Execute and verify changes.
3. Output a text page to the output file.
4. List an edited page on the console terminal or line printer.

4.4 CALLING THE EDITOR

The commands CREATE and EDIT call and run the Editor. Type

    .CREATE filename.ex

or

    .EDIT filename.ex

CREATE is used to open a new file for the first time, that is, create a completely new file. EDIT is used to open an already existing file that is to be modified or corrected. Thus, typing either command in response to the monitor dot (.) followed by a carriage return will call the Editor for your use.

The two valid I/O specification options for the Editor are given in Table 4-1.

Table 4-1 Editor Options

Option	Meaning
/B	Convert two or more spaces to a TAB when reading from input device.
/D	Delete the old copy of the output file (if one exists) before opening the new output file on the device. If /D is not used, the old copy of the output file is not deleted until all data has been transferred to the new file by an E or Q command.

4.5 MODES OF OPERATION

It is important to know the difference between the Editor command mode and the text mode. In the command mode, the Editor interprets all input typed on the keyboard as commands to perform some operation or allow some operation to be performed on the text stored in the buffer. In text mode, the Editor interprets all typed input as text to replace, to be inserted into, or to be appended to the contents of the text buffer.

Immediately after being loaded into memory and started, the Editor is in the command mode, and the text buffer is empty.

When the Editor enters the text mode (through the use of the Insert, Change or Append commands), corrections, additions or insertions may be made to the text. Typing a form feed (CTRL/L combination) will terminate the text mode by instructing the Editor to return to command mode. The Editor answers by prompting with a number sign (#) indicating that it has returned to the command mode.

The RETURN key is important in both the command and text modes. In the command mode, pressing the RETURN key allows the Editor to execute the command just typed. In the text mode, pressing the RETURN key after typing a line of text causes the line of text to be entered into the text buffer. A typed line is not actually part of the buffer until it is terminated by the RETURN key.

Editor key commands and special characters and their effect in the two Editor modes are described in Sections 4.6 and 4.7.

4.6 EDITOR KEY COMMANDS

The key commands in Table 4-2 allow you to transfer between modes. The commands are produced by pressing the CTRL key and the appropriate character key at the same time.

Table 4-2 Editor Key Commands

Command	Mode in Which Used	Meaning
CTRL/C	Text and Command Modes	Returns control to the Keyboard Monitor. All text that has been edited is lost. CTRL/C should be used with utmost caution, since no output file will be stored.
CTRL/D	Text Mode	Stops the listing of text. Returns control to Command Mode.
CTRL/L	Text Mode	Returns the Editor to Command Mode.
CTRL/U	Text Mode	Typing CTRL/U while entering text from the keyboard causes all text typed so far in the current line to be deleted. A carriage return/line feed is generated and the line may be retyped. (The command is equivalent to typing the DELETE key back to the beginning of the line.)
	Command Mode	When used in command, mode, CTRL/U cancels the entire command. The Editor performs a CR/LF and remains in the command mode (equivalent to DELETE key).

4.7 SPECIAL CHARACTERS

4.7.1 DELETE Key

The DELETE key on the terminal keyboard is used in error recovery in both the command and text modes. In the text mode, typing the DELETE key erases the last typed character. Repeated deletions erase the characters from right to left up to, but not including, the CR/LF, which separates the current line from the previous one. For example,

```
WRITE(6,25) (DEL) (DEL) (DEL) 15)
```

will be entered in the buffer as

```
WRITE(4,15)
```

When used in command mode, DELETE is equivalent to a CTRL/U combination and cancels the entire command. The Editor then prints a question mark (?), performs a CR/LF, and waits for another command to be typed.

4.7.2 Period (.)

The Editor assigns an implicit decimal number to the line on which it is currently operating. At any given time the period, which represents this decimal number, may be used as an argument to a command. In the following example, the L command is used since it allows text to be listed. Typing

```
*.L
```

means list the current line. Typing

```
*.-1,.+1L
```

means list the line preceding the current line, the current line, and the line following it, and then update the current line counter to the decimal number of the last line printed. The current line counter, represented by the period, is generally updated as follows:

1. After an R (Read page) or A (Append) command, the period is equal to the number of the last line in the buffer.
2. After an I (Insert) or C (Change) command, the period is equal to the number of the last line entered.
3. After an L (List) or S (Search) command, the period is equal to the number of the last line listed.
4. After a D (Delete) command, the period is equal to the number of the line immediately after the deletion.
5. After a K (Kill) command, the period is equal to 0.
6. After a G (Get and list) command, the period is equal to the number of the line displayed by the G.
7. After an M (Move) command, the period is not updated and remains whatever it was before the command.

4.7.3 Slash (/)

The symbol slash (/) has a value equal to the decimal number of the last line in the buffer. It may also be used as an argument to a command. For example,

```
*10,/L
```

means list from line 10 to the end of the buffer.

4.7.4 LINE FEED Key

Typing the LINE FEED while the Editor is in the command mode is equivalent to typing

```
.+1L
```

and will cause the Editor to display the line following the current one and to increment the value of the current line counter (dot) by one. LINE FEED does not perform this function while in the text mode.

4.7.5 Right-Angle Bracket (>)

Typing the right-angle bracket (>) while in command mode is equivalent to typing

```
.+1L
```

and will cause Editor to echo > and then display the line following the current line. The value of the current line counter is increased by one so that it refers to the last line displayed.

4.7.6 Left-Angle Bracket (<)

Typing the left-angle bracket (<) while in command mode is equivalent to typing

```
.-1L
```

and will cause Editor to echo < and then print the line preceding the current line. The value of the current line counter is decreased by one so that it refers to the last line printed.

4.7.7 Equal Sign (=)

Use the equal sign in conjunction with either the line indicator period (.) or slash (/). When typed in the command mode it causes the Editor to display the decimal value of the argument preceding it. In this way the number of the current line may be found (.=nnnn), or the total number of lines in the buffer (/=nnnn).

4.7.8 Colon (:)

The colon performs exactly the same function as the equal sign (=).

4.7.9 Tabulation (TAB)

The Editor simulates tab stops at eight space intervals across the terminal screen. When the TAB key is typed, the Editor produces a tabulation. A tabulation consists of from one to seven spaces, depending on the number needed to bring the carriage to the next tab stop. Thus, the TAB key produces neat columns and increases the legibility of programs and other text.

4.7.10 ESCape Key

The ESCape key is used to signal an intrabuffer character search when in the Editor command mode. Where typed, it echoes as a dollar sign ($) on the terminal screen. In the Editor text mode, typing the ESCape key echoes as a dollar sign, but it is stored in the file as an ESCape character (033).

4.7.11 Lower Case Characters

Lower case characters can be entered into a file in the Editor text mode. Releasing the keyboard CAPS LOCK key will enter all alphabetic characters in lower case. Upper case character can then by typed by pressing either SHIFT key. Locking the CAPS LOCK key will again generate all upper case characters.

4.8 EDITOR COMMANDS

Commands to the Editor are grouped under five general headings: Input, List, Output, Editing and Search. Explanation of the five kinds of commands is given in Sections 4.8.1 through 4.8.5. Error messages generated while using the Editor are discussed in Section 4.13.

The Editor contains an automatic text collector that reclaims buffer space following the use of a D(delete), S(search), or C(change) command. The text collector removes the deleted text and updates the necessary pointers. If a full buffer condition is reached, lines of text may be output using the P command, for example, and then these lines can be deleted from the buffer.

A command directs the Editor to perform a desired operation. Each command consists of a single letter, preceded by up to two arguments. The letter indicates the command function; the arguments usually specify which numbered line or lines of text are affected. Commands to the Editor must take one of the following forms, where X represents any command letter.

The commands discussed in the following pages can each be given whenever the Editor prints a number sign (#) at the left margin, Editor commands are given in upper case characters only. These commands are of the general form

```
#X
#nX
```

or

```
#m,nX
```

where m and n represent the line number designation (m must always be less than n), and X represents the command letter. Editor commands use decimal numbers.

NOTE

Except where specified, each line must be terminated by pressing the RETURN key.

4.8.1 Input Commands

Input commands allow text to be entered into the text buffer from the terminal.

Command	Format	Meaning
A	#A	Append the following text being typed at the keyboard until a form feed (ASCII 214 or CTRL/L) is found. The form feed returns control to command mode. Text input following the A command is appended to whatever is presently in the text buffer.
I	#I	Insert whatever text is typed before line 1 of the text buffer. The form feed (CTRL/L) terminates the insertion process and returns control to the command mode.

NOTE

Special characters, including lower case letters may be input to the file. The ESCape character is echoed as a dollar sign ($) for readability.

Command	Format	Meaning
	#nI	Insert whatever text is typed (until a form feed is typed) before line n of the text buffer.
R	#R	Read one page from the input device specified to the EDIT or CREATE commands and append the new text to the current contents of the buffer. If no input file was indicated or if no input remains, a question mark (?) is printed and the Editor returns to the command mode.

NOTE

In these commands, the Editor ignores ASCII codes 340 through 376. These codes include the codes for the lower case alphabet (ASCII 341-372). The Editor returns to the command mode only after the detection of a form feed or when the text buffer becomes full.

4.8.2 List Commands

List commands display all or any part of the contents of the text buffer on the terminal to permit examination of the text.

Command	Format	Meaning
L	#L	List entire contents of the text buffer on the terminal.
	#nL	List line n of the text buffer on the terminal.
	#m,nL	List lines m througn n of the text buffer on the terminal.

The Editor remains in command mode after a list command and the value of the current line counter is updated to be equal to the number of the last line printed.

Command	Format	Meaning
G	#G	Get and list the next line that has a label associated with it. A label in this context is any line of text that does not begin with one of the following: space (ASCII 240) / (ASCII 257) TAB (ASCII 211) RETURN (ASCII 215) At the termination of a G command, control returns to the command mode with the current line counter equal to the line just listed.
	#nG	Get and list the first line that begins with a label, starting the search at line n.
B	#B	Print the number of available memory locations in the text buffer. The Editor returns the number of locations on the next line. To estimate the number of characters that can be accommodated in this area, multiply the number of free locations by 1.7.

4.8.3 Output Commands

The format and meaning of the output commands are as follows:

Command	Format	Meaning
E	#E	Output the current buffer and transfer all remaining pages of input to the output file, close the output file and enter it in the directory. When this buffer is full, the text is output to the indicated output file. The P command automatically outputs a form feed after the last line of output, and returns control to the Monitor.

NOTE

If the E command is not used to close a file after editing, any changes, additions or corrections will not appear in the output file. Thus, the E command should usually always be the last command used in an editing session (also see Q command).

Command	Format	Meaning
P	#P	Write the entire text buffer to the output file.
	#nP	Write line n of the text buffer to the output file.
	#m,nP	Writes lines m through n, inclusive, to the output file.

NOTE

The P command automatically appends a form feed to its output, thus producing a page of text. This command allows you to paginate your listing. However, if the K command is not used after a P command, the text remains in the buffer and is again output with the new text read in before the next P command, thus resulting in an additive effect.

Command	Format	Meaning
K	#K	Kill the buffer. All text is deleted from the text buffer.

NOTE

The Editor ignores the commands nK or m,nK to prevent the buffer from accidentally being destroyed if the user means to type a List command (m,nL).

Command	Format	Meaning
Q	#Q	Immediate end-of-file. The Q command causes the text buffer to be output. The file is then closed (entered into the directory with the current date as its creation date); returns to the Monitor.
N	#N	Write the current buffer to the indicated output file and read the next logical page. The N command is equivalent to a P, K, R command sequence.
	#nN	Write the current buffer to the output file, kill the buffer, and read the next logical page. This is done n times until the nth logical page is in the text buffer. Control then returns to command mode. The N command cannot be used with an empty text buffer since there is no text to be written. If the buffer is empty when the N command is attempted, a question mark (?) is printed. For example, to read in the fourth page of a file, give the commands #R (to read the first page) and #3N (to read three more pages)
V	#V	The V command causes the entire text buffer to be listed on the line printer. The V command only works with the LA78 line printer. It does not work with the LQP78 line printer.
	#nV	List line n of buffer on the line printer.
	#m,nV	List lines m through n, inclusive, on the line printer.

4.8.4 Editing Commands

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

Command	Format	Meaning
C	#nC	Change the text of line n to the line(s) typed after the command is entered (typing a form feed terminates the text input). The C command is equivalent to a D command followed by an I command.
	#m,nC	Delete lines m through n and replace with the text line(s) typed after the command is entered. (Typing CTRL/L indicates the end of the changed lines.) The C command utilizes the text collector in altering text.
D	#nD	Delete line n from the buffer.
	#m,nD	Delete lines m through n from the buffer.
Y	#nY	Read (Yank) in n pages from the input file into the text buffer, without writing any output. For example, #5Y reads through four logical pages of input, deleting them without producing output. The fifth page is read into the text buffer and control automatically returns to command mode.

NOTE

This command should be used with caution, since it irrevocably deletes the contents of the text buffer.

Command	Format	Meaning
M	#m,n$pM	Move lines m through n directly before line p in the text buffer. The $ character represents the dollar sign and is typed using the dollar sign key (SHIFT/4). The old occurrence of the moved text is removed. This command can move one line but it needs three arguments. This is done by specifying the same line number twice. For example, #6,6$21M moves line 6 in front of line 21.

4.8.5 Search Commands

The search commands provide the ability to set the current line indicator designated by a period (.) in order to make changes at specific lines.

Command	Format	Meaning
S	#S	Character search command (Section 4.11.1).
J	#J	Interbuffer search command for character strings (Section 4.11.2).
F	#F	Follows a string search. Look for next occurrence of the string currently being sought. (Section 4.11.2).
ESC($)	#$TEXT' #'	Perform a intrabuffer character string search for the string TEXT. Following a string search, #' causes a search for the next occurrence of the string.

4.9 CREATING A NEW FILE

Suppose you want to create a simple FORTRAN program that will compute and print the powers of two. The following program will be used as an example.

```
C     FORTRAN DEMONSTRATION
C     COMPUTE AND PRINT POWERS OF TWO
      DIMENSION A(16)
      WRITE(4,15)
15    FORMAT(1H ,'POWER OF TWO')
      DO  20 N=1,16
      A(N)=2.**N
20    CONTINUE
      WRITE (4,25) (N,A(N),N=1,16)
25    FORMAT(1H ,'2** ',I2,'=',F10.1)
      STOP
      END
```

Use CREATE to open a new file called POWER.FT. Type

```
.CREATE POWER.FT
```

The Editor will then indicate that it is ready to accept an editing command by printing a number sign (#).

Now type the A command (an I command could also be used) to append text onto the empty text buffer:

```
#A
```

Start typing the program, using the TAB key to make your program easy to read. At the end of every line, press the RETURN key. After typing the program, return to the Editor command mode by typing a CTRL/L. Most editor commands automatically go back to the command mode to allow the next editor command to be typed in. However, the I(Insert), A(Append) and C(Change) commands (classified as text mode commands) require that a CTRL/L be typed to return to the command mode. A final command required to close the file and return to the Monitor is the E(Exit) command. The execution of the E(Exit) command closes out the file and places it on the device (DSK:). Verify this by typing the OS/78 command DIRECT, which displays the program just created in the directory listing. Note that the new file is given a creation date only when the date is entered using the OS/78 DATE command. The date must be reentered every time the system is booted up.

4.10 PAGING FILES

To paginate a file, either at a convenient point for dividing a program, or when the capacity of the text buffer is approached, transfer that part of the program already typed and clear the text buffer for the next portion of your program. Do this by using the P(Paging) command. Typing #P writes the entire text buffer to the output file and

places a FORM FEED character after the last line of text in the output buffer. The FORM FEED character allows the Editor to distinguish between pages of text. Before typing in the next page, be sure that there is no text in the buffer. This is done by using the K(Kill) command, which deletes any text in the buffer. Then type the A(Append) command, and continue typing your program. Upon completion, type the E(Exit) command to output the current buffer, transfer all input to the output file, and close the output file.

4.11 SEARCHING THROUGH FILES

Two types of searches are available to provide the ability to move within a program to insert changes and delete portions of a program. The first is the standard character search that searches for a single character (Section 4.11.1), and the second is the character string search that searches for a combination of characters (Section 4.11.2),

4.11.1 Single Character Search

The single character search command is one of the most useful functions in the Editor. It is also structured somewhat differently from other Editor commands.

The single character search is of the form

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

where m and n represent line numbers (m must always be less than n) and S initiates the search. This search command sets up to search the entire text buffer or the line or lines specified by the command.

4.11.1.1 nS Command – The nS command searches line n for the character specified after pressing the RETURN key which enters the command. The line, up to and including the character being searched for, is then displayed, output stops and all or any combination of the following operations may be undertaken. At this point the following options are available.

1. Delete the entire portion not yet displayed and terminate the line and the search by pressing the RETURN key.
2. Delete from right to left one of the printed characters each time the DELETE key is typed.
3. Insert characters after the last one printed simply by typing them.
4. Insert a carriage return/line feed, thus dividing the line into two, by pressing the LINE FEED key followed by CTRL/L.
5. Continue searching the line to the next occurrence of the search character by typing CTRL/L. When printing stops all options are again available.
6. Change the search character in the line and continue searching by typing CTRL/G(BELL) followed by the new search character. This allows all editing to be done in one pass.
7. Type CTRL/G(BELL) twice to terminate the command.

NOTE

The usual form of the character search command is #.S, followed by the RETURN key and then typing the desired character to be located. This form of the command is used to modify the current line.

4.11.1.2 m,nS Command – The m,nS command searches lines m through n inclusive in the same way as described above. The only difference is that pressing the RETURN key deletes the entire undisplayed portion of the line and terminates that line, but the search continues on the next line.

4.11.1.3 S Command – By typing S with no arguments, the entire buffer may be searched for occurrences of a single character, as above.

4.11.2 Character String Search

The character string search can identify a given line in the buffer by the presence in that line of any unique combination of characters. This search returns the line number as a parameter that can be used to further edit the text. Two types of string search are available: intrabuffer search and interbuffer search.

4.11.2.1 IntraBuffer Character String Search – The intrabuffer search scans all text in the current buffer for a specified character string. If the string is not found, a question mark (?) is displayed and control returns to the Editor command mode. If the string is found, the number of the line that contains the string is put into the current line indicator (.) and the Editor waits for another command to be issued. A command that uses this number may also be used.

Thus, searching for a character string in this manner furnishes a line number that can be used in conjunction with other Editor commands. This provides a useful framework for editing since it eliminates the need to count lines or search for line numbers by listing lines.

An intrabuffer search is signaled by pressing the ESCAPE key (which echoes as a dollar sign) in response to the Editor's number sign (#). The string to be found is then typed. The string can be up to 20 characters long. The search string cannot be broken across line boundaries, that is, it cannot contain a carriage-return/line-feed combination. Typing a single quote (') terminates the character string and causes the search to be performed beginning at line 1 of the text buffer. Use of the double quote (") instead causes the search to begin at the current line +1. (Use of the single quote (') and double quote (") as command elements prohibits their use in the search string.)

The FORTRAN program previously created will be used as an example in illustrating the features of character string search. First, open the file by typing

```
.EDIT POWER.FT
```

Then, in response to the number sign (#) sign, type

```
#R
```

which instructs the Editor to read in the program into the text buffer and return to the command mode. Typing

```
#L
```

displays the program on the terminal. The Editor again returns to the command mode (#). To list the line that contains WRITE, type

```
#$WRITE'L
```

where $ represents the ESCape key.

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

```
WRITE (4,15)
```

Control returns to the command mode, awaiting further commands. The next reference to WRITE can be found by typing

```
#"L
```

In this case, the double quote (") is a command that causes the last string searched for to be used again, with the search beginning at the current line +1. It is not necessary to enter the search string again. The command may be used several times in succession. For example, in a larger program the fourth occurrence of a string containing the characters WRITE is found by typing

```
#$WRITE'"""L
```

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

The properties of the commands single quote (') double quote (") allow for easy and useful editing, as the following example illustrates. To change DIMENSION A(16) to DIMENSION A(20) type the following commands:

```
#$FORTRAN'$DIM"C
(TAB)    DIMENSION A(20)
```

These instructions first cause the Editor to start at line 1 and search for the line beginning with FORTRAN. A search is then made for DIM starting from the line after the line containing FORTRAN. When this string is found, the line number of the line containing the string DIM becomes the current line number. The C(Change) command allows the line to be changed to the correct statement. Control is returned to the Editor command mode by typing CTRL/L. Use the TAB key to line up new or changed text that is being input to match the format that was originally used.

Since this search feature results in a line number, any operations that can be done by explicitly specifying a line number can be done by specifying a string instead. For example,

```
#$FORTRAN'+5L
```

will list the fifth line after the first occurrence of the text FORTRAN in the text buffer

```
DO 20 N=1,16
```

Typing

```
#$FORTRAN',$DO"L
```

will list all lines between the two labels, inclusive.

To list the line in which the format statement 25 appears the second time, type

```
#$25"L
```

The single quote (') accounts for the first 25 encountered in the WRITE statement. The double quote (") then seeks the second line that contains the number 25.

When both strings and explicit numbers are used, strings should be used first. For example, the commands

```
#1+DO'L
```

will not list the next line after the string DO occurs. The correct syntax is

```
#$DO'+1L
```

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

```
#J
$WRITE'
#.=0004
```

The J command initiates an interbuffer search. The Editor automatically displays the dollar sign ($), and then the character string that is sought is typed-in and terminated by the apostrophe. The search proceeds, and when the string is found, control returns to the command mode. Now type the .= construction to find the number of the line in the current buffer on which the string is contained, or any other desired editing commands.

To find further occurrences of the string WRITE, type the F command. The F command uses the last character string entered to search the buffer starting from the current line count + 1:

```
#F

#.=0008
```

This example causes a search for the string WRITE starting at the current line + 1. If no output file has been specified, the J or F command read the next input buffer without attempting to produce any output.

NOTE

Use the J command for interbuffer searches only. After the J or F command has processed the entire input file, execute either an E or Q command to close the output file.

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

Command	Explanation
CTRL/U	A CTRL/U will return control to the Editor command mode if executed while entering text in a string search command; the string search command is ignored, as in the following example:

```
#J
$DIM ^U
#
```

The interbuffer search for the characters DIM was aborted by the user typing ^U before terminating the string with single quote (') or double quote (").

Command	Explanation
DELETE	Pressing the DELETE key while entering text for a string search causes the text so far entered to be ignored and allows a new string to be inserted. The Editor displays a dollar sign ($) in response to the DELETE key, as shown in the following example:

```
#$WRITE (DEL)
$
```

4.12 EDITING AN EXAMPLE PROGRAM

It is often necessary to edit a closed file, that is, to change the code, correct overlooked errors, or insert additional information.

This section illustrates how to use the various editing commands in editing an already existing assembly language program. The example program is shown below. It is written in PAL8 assembly language and when executed displays HELLO!. Note that the example has the errors deliberately left in it.

```
        *200
        MONADR=7600
START,  CLA CLL         /CLEAR ACCUMULATOR AND LINK
        TLS             /CLEAR TERMINAL FLAG
        TAD BUFADR      /SET UP POINTER
        DCA PNTR        /FOR GETTING CHARACTERS
NEXT,   TFF             /SKIP IF TERMINAL FLAG SET
        JMP .-1         /NO: CHECK AGAIN
        TAD I PNTR      /GET A CHARACTER
        TLS             /PRINT A CHARACTER
        ISZ PNTR        /DONE YET?
        CLA CLL         /CLEAR ACCUMYLATOR AND LINK
        TAD I PNTR      /GET ANOTHER CHARACTER
        SZA CLA         /JUMP ON ZERO AND CLEAR
        JMP I MON       /RETURN TO MONITOR
        JMP NEXT        /GET READY TO PRINT ANOTHER
BUFADR  BUFF            /BUFFER ADDRESS
PNTR,   BUFF            /POINTER
BUFF,   215;212;"H;"E;"L;"L;"O;"!;0
MON,    MONADR          /MONITOR ENTRY POINT
```

You have checked the above program and want to make the following changes:

1. Insert a comment line at beginning of program.
2. Correct TFF to TSF in line with label NEXT.
3. Correct the spelling of accumulator where it appears the second time.
4. Transpose lines JMP I MON and JMP NEXT.
5. Add a comma after label BUFADR.

Typing the EDIT command and the file called SAMPLE.PA will open that file and cause the Editor to prompt with the number sign (#) thereby indicating it is ready to receive the editing commands:

```
.EDIT SAMPLE.PA
#
```

If you are changing a part of a program, but want to save your original file, type the following commands:

```
.RENAME SAMPLE.BK<SAMPLE.PA
.EDIT SAMPLE.PA<SAMPLE.BK
```

The file that is changed will be output as SAMPLE.PA, while SAMPLE.BK will remain unchanged.

At this point, the text buffer is empty. Type the R (Read) command to read in the first page of the file, and in the case of this example, the entire program. After this command is executed, the Editor prompts with the number sign (#) for another command. Use the L command to display the program.

The first change is the addition of the comment line. Simply type the I(Insert) command and the text to be inserted as follows:

```
#I
/ROUTINE TO TYPE A MESSAGE
```

This command inserts whatever text is typed before line 1 of the text buffer, and in this example, becomes line 1. The RETURN key is typed after the line to allow the Editor to recognize this text as a separate line. CTRL/L is then typed to return to the Editor command mode. Verify that this is now line 1 by typing the command

```
#1L
```

which displays:

```
/ROUTINE TO TYPE A MESSAGE
```

Your next change is that of correcting TFF to TSF.

Use the string search ESCape command to find and display the line to be corrected as follows:

```
#$TFF'L
```

Use the single character search option to make the correction in the line by typing.

```
#.S
```

followed by a RETURN key.

The line number can be left out since the period is equal to the number of the last line searched for the L command. Now type the search character which is F. The line will be displayed up to the F. Delete the F by pressing the DELETE key and type in the correct character S. Since this completes all changes in this line, type CTRL/G (BELL) followed by another CTRL/G (BELL) to return to the command mode.

The next change to be made is the correction of the word ACCUMYLATOR to ACCUMULATOR. This example will illustrate the ability of the search commands to accomplish repetitive searches for identical character strings. The intrabuffer search commands are shown here to illustrate their use.

The intrabuffer search uses the ESCape key and the single quote (') and double quote (") constructions.

Typing

```
#$CLA'"L
```

will list the line containing the second occurrence of CLA, that is,

```
CLA CLL   /CLEAR ACCUMYLATOR AND LINK
```

Note that the incorrect word still appears. The word can be corrected by typing the single character search command S and using the DELETE key. The period construction is used with the S command since the period is equal to the number of the last line listed.

Type

```
#.S
```

Then type a Y, the character that is to be searched for. Use the DELETE key to erase this character and type the correct character U.

Since this concludes the corrections on this line, get back to the Editor command mode by typing CTRL/G twice. Note that if only a carriage return were typed after making the correction, control would return to the command mode, but the remainder of the line after the corrected character to the end of the line would be deleted.

Verify that the proper correction was made by typing

```
#.L
```

which will list the corrected line

```
CLA CLL    /CLEAR ACCUMULATOR AND LINK
```

The next correction is to transpose the two lines JMP I MON and JMP NEXT.

Use the character string search command J since this command gives the number of the line to be moved. Type

```
#J
$JMP I MON'
#.=0016
```

Use the M(Move) command to transpose the line by typing

```
16,16$18M
```

This command moves line 16 (JMP I MOV) before line 18, or after line 17 (JMP NEXT), which in effect performs the transposition. It also deletes the initial occurrence of the moved line, now making JMP I MON the 17th line. Type 17L to verify this.

The next correction requires the addition of a comma after the label BUFADR. The G(Get) command allows you to get the next line that has a label associated with it after the line that is equal to the current line counter. Thus, typing

```
#G
```

will get the line starting with BUFADR and display it on the terminal as follows:

```
BUFADR     BUFF     /BUFFER ADDRESS
```

Then make the correction by adding a Comma after the label, using the change or character search commands previously described.

When all the changes have been made, display the corrected file on the terminal. Type

```
#L
```

and the entire contents of the text buffer will be displayed. Remember to close out the file and return to the Monitor by typing the E command.

During the editing of a file, the output device specified in the command string may become full before the editing process is complete.

If this happens and a write is attempted on that device, an error occurs. The output file is closed, and the following message is displayed:

```
FULL
*
```

A new output device and file must be indicated before continuing any further editing. Since the contents of the text buffer are retained through this procedure, no text will be lost if this error occurs.

NOTE

An output file must be supplied following the asterisk (*), and terminated with a left-angle bracket (<) and the RETURN key. However, specifying an improper output device will cause a fatal error that will destroy the output file.

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

```
.EDIT OUT<IN
*Y
*J
$STRING'
FULL
*RXA1:OUT2<
```

Device DSK: is full; RXA1: is specified as the new output device, and editing continues.

```
*.L  TAD STRING
*.D
*E
FULL
*RXA1:OUT3<
```

Device RXA1: has become full; remove and insert new diskette. RXA1: is specified as the output device, and editing continues.

At this point, the output "file" is the series of files – DSK:OUT, RXA1:OUT2, and RXA1:OUT3. When output is split like this, the split may have occurred in the middle of a line. Therefore, the output files should never be edited separately since the split lines will then be lost.

4.13 EDITOR ERROR MESSAGES

Two types of error messages, minor and major, are generated when an error is made while running the Editor.

Minor errors, such as incorrect format in a command string or a search for nonexistent information, cause the Editor to display a question mark. For example, if a command requires two arguments, and only one is provided, the Editor will display a question mark (?), perform a carriage return/line feed, and ignore the command as typed. Similarly, if an illegal or unrecognized command character is typed, the error message ? will be displayed, followed by a carriage return/line feed; the command will be ignored. However, if an argument is provided for a command that does not require one, the argument may be ignored and the normal function of the command performed. The following examples illustrate minor errors that can be encountered while using the Editor.

Message	Explanation
L ?	The buffer is empty. Nonexistent information is requested.
7,5L ?	The arguments are in the wrong order. The Editor cannot list backwards.

Message	Explanation
17$10M ?	This command requires two arguments before the $; only one was provided.
H ?	Nonexistent command letter.

Major errors cause control to return to the Monitor and may be due to one of the causes listed in Table 4-3. These errors cause a message to be printed in the form

?n ^C

where n is an error code listed in the table and ^C indicates that control has passed to the Monitor. These errors will generally result in complete loss of the output file.

Table 4-3 Editor Error Codes

Error Code	Meaning
0	Editor failed in reading a device. Error occurred in device handler; most likely a hardware malfunction.
1	Editor failed in writing onto a device; generally a hardware malfunction.
2	File close error occurred. For some reason the output file could not be closed; the file does not exist on that device.
3	File open error occurred. This error occurs if the output device is a read-only device or if no output file name is specified on a file-oriented output device.
4	Device handler error occurred. The Editor could not load the device handler for the specified device. This error should not normally occur.

4.14 SUMMARY OF EDITOR COMMANDS AND SPECIAL CHARACTERS

The command and special characters discussed in this chapter are summarized in Table 4-4.

Table 4-4 Editor Command and Special Characters

Command	Format	Meaning
A	#A	Append the following text being typed at the keyboard until a CTRL/L (form feed) is typed. The form feed returns control to the command mode. Text input following the A command is appended to whatever is present in the text buffer.
B	#B	List the number of available memory locations in the text buffer. The Editor returns the number of locations on the next line. To estimate the number of characters that can be accommodated in this area, multiply the number of free locations by 1.7.
C	#nC	Change the text of line n to the line(s) typed after the command is entered (typing a CTRL/L terminates the input).

Continued on next page

Table 4-4 (Cont.) Editor Commands and Special Characters

Command	Format	Meaning
	#m,nC	Delete lines m through n and replace with the text line(s) typed after the command is entered. (Typing CTRL/L indicates the end of the inserted lines.)
D	#nD	Delete line n from the buffer.
	#m,nD	Delete lines m through n from the buffer.
E	#E	Output the text buffer and transfer all remaining pages of the input file to the output file, closing the output file and returning to the Monitor.
F	#F	Follows a string search. Look for next occurrence of the string currently being sought (by the J command).
G	#G	Get and list the next line that has a label associated with it. A label in this context is any line of text that does not begin with one of the following: space (ASCII 240) / (ASCII 257) TAB (ASCII 211) RETURN (ASCII 215) At the termination of a G command, control goes to the command mode with the current line indicator (.) equal to the line just listed.
	#nG	Get and list the first line which begins with a label, starting the search at line n.
I	#I	Insert whatever text is typed before line 1 of the text buffer. Typing CTRL/L terminates the entering process and returns control to the Editor command mode.
	#nI	Insert whatever text is typed (until a CTRL/L is typed) before line n of the text buffer.
J	#J	Interbuffer search command for character strings (see Section 4.11.2 describing the InterBuffer Character String Search).
K	#K	Kill the buffer. Delete all text from the text buffer.
NOTE The Editor ignores the commands nK and m,nK to prevent the buffer from accidentally being destroyed if the user means to type a List command (m,nL).		

Continued on next page

Table 4-4 (Cont.) Editor Commands and Special Characters

Command	Format	Meaning
L	#L	List entire contents of the text buffer on the terminal.
	#nL	List line n of the text buffer on the terminal.
	#m,nL	List lines m through n of the text buffer on the terminal. Control then returns to command mode.
M	#m,n$pM	Move lines m through n directly before line p in the text buffer. The $ character represents typing the dollar sign key (SHIFT/4). The old occurrence of the moved text is removed.
N	#N	Write the current buffer to the output file and read the next page.
	#nN	Write the current buffer to the output file, kill the buffer, and read the next page. This action is done n times until the nth page is in the text buffer. Control then returns to command mode. The N command cannot be used with an empty text buffer. A question mark (?) is printed if this is attempted.
P	#P	Write the entire text buffer to the output file. The P command automatically outputs a FORM character (214) after the last line of output.
	#nP	Write line n of the text buffer to the output file and a FORM character.
	#m,nP	Write lines m through n, inclusive, to the output file and a FORM character.
Q	#Q	Immediate end-of-file. Q causes the text buffer to be output and the file closed.
R	#R	Read one page from the input device and append the new text to the current contents of the text buffer. In no input file was indicated or if no input remains, a question mark (?) is displayed and control returns to the command mode.
S	#S	Character search command (see Section 4.11.1).
V	#V	The V command causes the entire text buffer to be listed on the line printer.
	#nV	List line n of the text buffer on the line printer.
	#m,nV	List lines m through n, inclusive, on the line printer.
Y	#nY	Read (Yank) in a logical page from the input file, without writing any output. For example, #5Y

Continued on next page

Table 4-4 (Cont.) Editor Commands and Special Characters

Command	Format	Meaning
		reads through four logical pages of input, deleting them without producing output. The fifth page is read into the text buffer and control automatically returns to the command mode.
$(ESC)	#$TEXT' #"	Perform a character string search for the string TEXT. Following a string search, #" causes a search for the next occurrence of the string (see Section 4.11.2 describing the IntraBuffer Character String Search).
.=or.: /=or/:		Typing these characters obtains the current line number (.=) and the last line number in the text buffer (/=). The number is printed by the Editor immediately after the user types the equal sign. (The colon character is equivalent to the equal sign.)
>	#>	Equivalent to .+1L, list the next line in the text buffer.
<	#<	Equivalent to .-1L, list the next line in the text buffer.
LINE FEED Key		Equivalent to .+1L, list the next line in the text buffer.
#	##	Print the current Editor version number.

NOTE

If the search fails when using the F or J command, further commands will cause the system to prompt with a ?. The file must be closed, and then reopened again.

CHAPTER 5
THE PAL8 ASSEMBLER

5.1 INTRODUCTION

PAL8 is the OS/78 Operating System assembler. It is used to generate binary object files from source programs written in the PAL8 assembly language. Once a source program is edited and stored on a diskette, it is ready for assembly.

PAL8 is a two-pass assembler. The source program is read in pass 1 and an internal symbol table is produced. Any new symbols used in the source program are defined and added to the PAL8 permanent symbols during pass 1. During pass 2, the assembler reads the source file again and generates the binary code using the symbol table definitions created during pass 1 and continues defining symbols as well. The binary file that is output may be loaded into memory as the "current" executable program by the LOAD command. Absolute binary format consists of 8-bit bytes, containing field setting commands, address setting commands, and sequential data words. An optional third pass may be involved if a program listing is desired. During pass 3, the assembler reads the source file a final time and generates the assembly listing as an ASCII (character string) file. The listing consists of the source statement together with its current location counter and the generated code in octal. The first 40 (decimal) characters of the first line of each page of the listing contains a title, the assembler version number, the date and the listing page number.

Use the OS/78 command PAL to call the assembler. It can also be called by the commands CREF and EXECUTE; use of these commands is explained in this chapter.

The PAL command specifies the binary and listing output devices and file names, the input devices and file names, and any options selected by the user. From one to nine input files may be specified. The typical way to assemble, load, and then run a program called PROG is as follows:

```
.PAL PROG              – Assemble the program
.LOAD                  – Load the program into memory
.SAVE SYS PROG         – Save the program
.R PROG                – Run the program
```

The long form of the command string is

```
.PAL DEV:BINARY,DEV:LISTING,DEV:CREFLS<DEV:INPUT,.../OPTIONS
```

If the extension to the file name is omitted, the following extensions are assumed

.PA for input files.
.BN for binary output file.
.LS for listing output file.
.TM for intermediate CREF file.

If an assembly or CREF listing is not desired, omit the listing file or CREF file, respectively.

For example, to assemble, load, and run a PAL8 program named SAMPLE, which is stored on diskette unit 1, type

```
.PAL RXA1:SAMPLE/G-T
```

After assembly the program is loaded and run (since the /G was specified) with the starting address assumed to be location 0200 in field 0; the binary file is stored on the DSK: device as SAMPLE.BN. The CCL option -T displays the assembled program listing on the terminal.

If a binary file is not desired, specify the -NB option at the end of the command line (NB stands for No Binary). For example, to get a listing only, type

```
.PAL SAMPLE-LS-NB
```

The -LS option indicates that a listing should be produced.

The assembler displays any error messages encountered in the program on the terminal, even when a listing is not produced. Typing CTRL/O at the keyboard during an assembly suppresses the display of error messages. However, messages are still printed in the listing file (if any) and occur immediately before the line that is in error.

For example, the command line

```
.PAL SAMPLE/S-LS
```

causes PAL8 to assemble SAMPLE.PA (or SAMPLE), generating DSK: SAMPLE.BN and putting the listing into the file SAMPLE.LS on the default device DSK. The /S option suppresses listing of the symbol table.

The command line

```
.PAL BIN<SAMPLE.PA/G=600
```

assembles SAMPLE.PA, putting the binary output into a file named BIN.BN, and then loading the file BIN.BN and starting it at 600. The construction =600 is an option that specifies the starting address.

Assembly can be terminated at any time by typing CTRL/C on the keyboard, and any output files being stored will be deleted. Otherwise, PAL8 always returns to the Monitor upon completion of assembly.

A source program may be broken down into multiple files. There files may then be assembled by specifying them as input files separated by commas. For example,

```
.PAL PART1,PART2,PART3
```

assembles a three-part program. This technique is useful when it is desired to assemble two programs that are identical except for a few lines at the beginning of the programs. There different lines can be broken out into a "prefix file". For example, two different file assemblies may be generated by

```
.PAL PRFX1,FILE
```

and

```
.PAL PRFX2,FILE
```

Up to nine input files may be concatenated together in an assembly specification.

If more than one input file is specified, and output files are desired but not explicitly specified, the name of the first input file is used for the output file names. For example,

```
.PAL A,B
```

produces the binary file A.BN.

If a file name other than the first input file is desired for the binary file name, use the -NB option after the last input file name not desired as the binary file name. For example,

```
.PAL A-NB,B
```

produces DSK: B.BN and

```
.PAL A,B,C-NB,D,E,F
```

produces DSK: D.BN.

If a -LS option is specified, it must appear immediately after an input file name. This is the name that will be used for the name of the listing file. For example,

```
.PAL A,B-LS
```

produces DSK: B.LS while

```
.PAL A-LS,B
```

produces DSK: A.LS.

A CCL -L or -T option used with a PAL or COMPILE command outputs the listing on the line printer and terminal, respectively.

Note that the PAL command normally produces a binary file even when a name is not given. Thus, typing

```
.PAL ,LPT:< file
```

produces a binary file.

The PAL command assumes the input file type (extension) is .PA when none is specified. Thus, the command

```
.PAL TEST
```

looks for a file DSK: TEST.PA. If no file with .PA is found on DSK:, then a file test without any extension is looked for next. It is good practice to include a .PA extension with any PAL8 source files to remind you what type of source file it is.

The COMPILE and EXECUTE commands may also be used to invoke PAL8. These commands search the directory of the specified device for the file given with the command, and if one is found with a .PA extension, PAL8 is invoked. For example,

```
.COMPILE TEST
```

will run PAL8 if TEST.PA is found. An unusual extension may be explicitly specified by typing

```
.PAL TEST.XX
```

which will assemble DSK: TEST.XX. To specify PAL8 as the processor in the COMPILE command, include -PA as a CCL option in the command line as follows:

```
.COMPILE TEST.XX-PA
```

The EXECUTE command is similar to the COMPILE command except that the EXECUTE command is supported by the /G option.

If an argument is not given with a PAL, or COMPILE, or EXECUTE command, the argument used with the last such command is assumed when that command is used again.

5.2 CREATING AND RUNNING AN ASSEMBLY PROGRAM

The following steps demonstrate the procedure for creating and running a PAL8 program.

5.2.1 Creating a Program

Create the assembly language source file by calling the Editor as follows:

```
.CREATE  SAMPLE.PA
```

Since a new program is being created, only a single file name need be specified. The OS/78 Editor will then display a number sign (#) to indicate it is ready to accept a command. (See Chapter 4 for a detailed discussion of the OS/78 Editor.)

Type the A (Append) command to allow the Editor to accept text. Then type in the program, one line at a time with each line followed by the RETURN key.

```
#A

/ROUTINE TO TYPE A MESSAGE
        *200
        MONADR=7600
START,  CLA CLL         /CLEAR ACCUMULATOR AND LINK
        TLS             /CLEAR TERMINAL FLAG
        TAD BUFADR      /SET UP POINTER
        DCA PNTR        /FOR GETTING CHARACTERS
NEXT,   TSF             /SKIP IF TERMINAL FLAG SET
        JMP .-1         /NO: CHECK AGAIN
        TAD I PNTR      /GET A CHARACTER
        TLS             /PRINT A CHARACTER
        ISZ PNTR        /DONE YET?
        CLA CLL         /CLEAR ACCUMULATOR AND LINK
        TAD I PNTR      /GET ANOTHER CHARACTER
        SZA CLA         /JUMP ON ZERO AND CLEAR
        JMP NEXT        /GET READY TO PRINT ANOTHER
        JMP I MON       /RETURN TO MONITOR
BUFADR, BUFF            /BUFFER ADDRESS
PNTR,   BUFF            /POINTER
BUFF,   215;212;"H;"E;"L;"L;"O;"!;0
MON,    MONADR          /MONITOR ENTRY POINT
```

Now type a CTRL/L to terminate input. This command returns you to the Editor command mode.

Type the L (List) command in response to the Editor's number sign (#) to list the text that was inserted into the text buffer.

When you are satisfied that the input is correct, type the E (Exit) command to store the file and return to the Monitor.

5.2.2 Assembling a Program

Now assemble the source program just created. Use the command:

```
.PAL SAMPLE-LS
```

This command creates a binary file SAMPLE.BN and produces a listing called SAMPLE.LS on the diskette (performed by the CCL -LS option). Use the TYPE command to display the listing on the terminal or the LIST command to print the listing on a line printer.

The assembly listing produced by PAL appears as follows:

```
/ROUTINE TO TYPE A MESSAGE                       PAL8-V10A   11-MAY-77   PAGE 1

                   /ROUTINE TO TYPE A MESSAGE
        0200                 *200
        7600                 MONADR=7600
00200   7300       START,    CLA CLL             /CLEAR ACCUMULATOR AND LINK
00201   6046                 TLS                 /CLEAR TERMINAL FLAG
00202   1216                 TAD BUFADR          /SET UP POINTER
00203   3217                 DCA PNTR            /FOR GETTING CHARACTERS
00204   6041       NEXT,     TSF                 /SKIP IF TERMINAL FLAG SET
00205   5204                 JMP .-1             /NO: CHECK AGAIN
00206   1617                 TAD I PNTR          /GET A CHARACTER
00207   6046                 TLS                 /PRINT A CHARACTER
00210   2217                 ISZ PNTR            /DONE YET?
00211   7300                 CLA CLL             /CLEAR ACCUMULATOR AND LINK
00212   1617                 TAD I PNTR          /GET ANOTHER CHARACTER
00213   7640                 SZA CLA             /JUMP ON ZERO AND CLEAR
00214   5204                 JMP NEXT            /GET READY TO PRINT ANOTHER
00215   5631                 JMP I MON           /RETURN TO MONITOR
00216   0220       BUFADR,   BUFF       /BUFFER ADDRESS
00217   0220       PNTR,     BUFF       /POINTER
00220   0215       BUFF,     215;212;"H;"E;"L;"L;"O;"!;0



00221   0212
00222   0310
00223   0305
00224   0314
00225   0314
00226   0317
00227   0241
00230   0000
00231   7600       MON,      MONADR              /MONITOR ENTRY POINT


/ROUTINE TO TYPE A MESSAGE                       PAL8-V10A   11-MAY-77   PAGE 2

BUFADR  0216
BUFF    0220
MON     0231
MONADR  7600
NEXT    0204
PNTR    0217
START   0200


ERRORS DETECTED: 0
LINKS GENERATED: 0
```

The first column of the listing gives the field number and octal address. The second column is the assembled object code. The symbol table is printed at the end followed by the number of errors detected and links generated. Link generation is described in Section 5.12. Each error generates on error message. The error messages are described in Section 5.14.

If errors have been detected, the program has been written or typed incorrectly. Check it again.

The COMPILE command may also be used to assemble the program by typing

```
.COMPILE SAMPLE
```

Several options are available with the PAL command. The options are described in detail in Section 5.3.

5.2.3 Loading and Saving a Program

Load the binary file generated by assembling SAMPLE.PA into memory by typing

```
.LOAD SAMPLE
```

The SAMPLE program is now the "current" memory image.

If more than nine input files are to be specified, specify the nine input files in the command line, then press the ESCape key. This causes the Command Decoder (see Appendix D) to print an asterisk (*). Then type the additional files in response to the asterisk, terminating each file by a RETURN key.

Since programs in memory image format can be executed directly, it is desirable to save this format of your program. Do this by typing

```
.SAVE SYS SAMPLE
```

The memory image fomat of SAMPLE is now both in memory and on the system device as a new file called SAMPLE.SV.

5.2.4 Executing the Program

Since the program now resides on the diskette and in main memory, it can be executed by typing

```
.START
```

Otherwise, the file SAMPLE.SV can be loaded into memory from SYS: and run by typing

```
.R SAMPLE
```

The program will be executed and HELLO! will appear on the screen.

Also, the program can be assembled, loaded and run by simply typing

```
.EXECUTE SAMPLE
```

This command produces the binary file SAMPLE.BN, loads it into memory, and starts it running.

Another method that is available to assemble, load and run a PAL8 assembly language program is the use of the /G option with the PAL command. Typing

```
.PAL SAMPLE/G
```

assembles the input file SAMPLE.PA, loads the binary file, and executes the program. Also, the command

```
.LOAD SAMPLE/G
```

will load the binary file SAMPLE.BN and execute it.

5.2.5 Getting and Using a Cross-Reference Listing

The Cross-Reference Program (CREF) is an aid in debugging assembly language programs by providing the ability to pinpoint all references to a particular symbol.

Generate the CREF listing by using the CREF command or the PAL command with the /C option. Typing

```
.PAL SAMPLE/C-LS
```

will produce a binary file and the CREF listing as a file called SAMPLE.LS. Using the TYPE or LIST command will display the listing on the terminal or print it on a line printer, respectively. Further information on CREF is given with the discussion of the CREF command in Chapter 3.

The output of CREF is identical to the PAL8 assembler output except that the CREF program numbers each line in decimal and generates a cross-reference table after the listing that has the following format:

```
BUFADR      6     18#
BUFF       18     19     20#
MON        17     29#
MONADR      3#    29
NEXT        8#    16
PNTR        7     10     12     14     19#
START       4#
```

The cross-reference table contains every user-defined symbol and literal, sorted alphabetically. If literals are used, each literal is indicated by an underline followed by the field and address at which the literal occurs. For each symbol and literal there appears a list of numbers that specify the lines in which each is referenced. The line where the symbol is defined is followed by a #.

5.2.6 Obtaining a Memory Map

Many times it is desirable to obtain a map of a program showing memory locations used by the given binary file. Generate the map by using the MAP command. Typing

```
.MAP SAMPLE-L
```

will print the map on a line printer from SAMPLE.BN, and typing

```
.MAP SAMPLE
```

will display the map on the terminal (in effect, equivalent to the command MAP SAMPLE -T).

The input file must always be a binary file (.BN extension). Use the command

```
.MAP MAPFIL<SAMPLE
```

to place the map in the file MAPFIL.MP. It can be displayed on the terminal or printed on a line printer by using the commands TYPE and LIST, respectively. The map for the program SAMPLE is shown below.

```
BITMAP V4    FIELD 0

      0000000011111111222222223333333344444444555555556666666677777777
      0123456701234567012345670123456701234567012345670123456701234567

00000
00100

00200 1111111111111111111111111100000000000000000000000000000000000000
00300

00400
00500

00600
00700
.
.
.
```

The output is a series of lines that are made up of a string of digits. Each digit represents a single memory location, and can have a value of 0 to 3. A 0 means that the location is empty while a 1 means that the location was loaded into into once. The appearance of a 2 or 3 means that a location was loaded into two (2) or three or more times (3) and may imply a programming error in that two or more separate routines are each trying to load values into the same location. The example program shows memory locations 0200 through 0231 being loaded into once which is correct. Further information on the MAP command is given in Chapter 3.

5.3 PAL8 OPTIONS

The command string typed for the PAL command may include several options. The options available are listed in Table 5-1.

Table 5-1 PAL8 Options

Option	Meaning
/B	This option makes the operator ! a 6-bit left shift instead of an inclusive OR (A!B equals A^100 B). This allows you to pack two 6-bit ASCII characters into a 12-bit word. This effect applies for the entire assembly.
/C	Run SYS:CREF.SV after assembly. The third output file specified (optional) is the temporary output file passed to CREF. The second output file is the listing file to be produced. If no third output file is given, SYS:CREFLS.TM is assumed and will be deleted after use. The /C option supersedes the /G and /L options if specified in the same command string.
/E	Enable error messages if a link is generated. The LG error message is generated as well as the link being flagged.

Continued on next page

Table 5-1 (Cont.) PAL8 Options

Option	Meaning
/F	Disable extra zero fill in TEXT pseudo-op. If the text in the TEXT pseudo-op contains an even number of characters, no word of zeros will be added to the end.
/G	Load the binary file into memory and begin execution at the indicated starting address. If no starting address is indicated, start at 200.
/H	Generate nonpaginated output. Headers (including page numbers and page format) are suppressed.
/J	Do not list lines containing code in conditional brackets which is conditionalized out.
/K	Used in assembling very large programs; allows more space in Field 1 to be used for symbol table storage.
/L	Load the resulting binary file into memory but do not start it.
/N	Generate the symbol table, but not the rest of the listing.
/O	Disable origining to 200 after a FIELD pseudo-op. The origin remains what it was before the FIELD pseudo-op.
/S	Omit the symbol table normally generated with the listing.
/T	Output a carriage return/line feed in place of form feed character(s) in the program listing.
/W	Do not remember the number of literals that were previously stored on a page after origining off page and then back on again.

When the /L or /G option is specified, you can also include any option in the command line for the LOAD command, such as = starting address option. If no address is specified, 00200 is assumed. If no binary output file is specified (by using the CLL -NB option) with a /L or /G, a temporary file SYS:PAL8BN.TM is created and loaded.

5.4 CHARACTER SET

PAL8 programs are composed of physical lines containing assembly language mnemonics that indicate processor instructions, user-defined symbols, comments, listing control characters and pseudo-operators (assembler directives). The following characters are used to specify these components.

1. The alphabetic characters A through Z.
2. The numeric characters 0 through 9.
3. The characters described in following sections as special characters and operators.
4. Characters that are ignored during assembly such as LINE FEED and FORM FEED.

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

IC nnnn

to be printed during passes 1 and 2 where nnnn represents the octal location at which the illegal character occurred. (As assembly proceeds, each instruction is assigned a location determined by the current location counter. When an illegal character or any other error is encountered during assembly, the value of the current location counter is displayed in the error message.)

5.5 STATEMENTS

PAL8 source programs are usually prepared on the terminal (using the Editor) as a sequence of statements. Each statement is written on a single line and is terminated by a carriage return. There are four types of elements in a PAL8 statement which are identified by the order of their appearance in the statement and by the separating (or delimiting) character that follows or precedes the element. These types are as follows:

1. label
2. instruction
3. operand
4. comment

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

5.5.1 Labels

A label is the symbolic name created by the programmer to identify the location of a statement in the program. If present, the label is written first in a statement. It must begin with an alphabetic character, contain only a alphanumeric characters, and be terminated by a comma; there must be no intervening spaces between any of the characters and the comma. A label may be of any length but only the first six characters are significant. If a label is the only element on a line, it identifies the location of the next program location. For example,

A,	Defines A as 00200
B,0	Defines B as 00200 and stores 0000 at location 00200

5.5.2 Instructions

An instruction may be one or more of the mnemonic machine instructions or a pseudo-operation that directs assembly processing. (Assembly pseudo-ops are described in Section 5.11.) Instructions are terminated with zero or more spaces (or tabs) followed by a semicolon, slash, or the end of the line.

5.5.3 Operands

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

5.5.4 Comments

Comments are arbitrary strings of characters that begin with a slash (/). Comments do not affect assembly processing or program execution but are useful in the program listing to record information for later analysis or debugging. The assembler ignores everything from the slash to the next carriage return.

It is possible to have only a carriage return on a line, resulting in a blank line in the final listing. Such a line is ignored; the current location counter is not incremented.

5.6 FORMAT CHARACTERS

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

5.6.1 Form Feed

The form feed character causes the assembler to output blank lines (or a form feed character if listing on the line printer) in order to skip to a new page in the output listing during pass 3; this is useful in creating a page-by-page listing. The form feed is generated by the Editor P(Page) command. The Pseudo-op EJECT may also be used to form pages in the assembly listing (see Section 5.11.7).

5.6.2 Tab

Tabs are used in the body of a source program to separate fields into columns. 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
```

Each occurrence of a tab character causes PAL8 to output enough spaces to move to the next text column. Each text column is 8 characters wide.

5.6.3 Statement Terminators

Each statement is terminated by the carriage return/line feed character combination produced by the Editor when the RETURN key was pressed during the Insert or Append modes. The semicolon (;) may also be used as a statement terminator and is considered identical to a carriage return except that it will not terminate a comment. For example,

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

The entire expression between the slash and the end of the line is considered a comment. Thus in this case the assembler ignores the TAD B. If, for example, a sequence of instructions to rotate the contents of the accumulator and link six places to the right is desired, it can be written as follows:

```
RTR
RTR
RTR
```

However, as an alternative, all three instructions can be placed on a single line by separating them with the special character semicolon and terminating the entire line with a carriage return. The above sequence of instructions can then be written

```
RTR;RTR;RTR
```

These multistatement lines are particularly useful when setting aside a section of data storage. For example, a 4-word block of data could be reserved by specifying either of the following:

```
LIST,       1;2;3;4
```

or

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

5.7 NUMBERS

Any sequence of digits delimited by either a SPACE, TAB, semicolon, or end of the line forms a number. PAL8 initially interprets numbers in octal (base 8). This can be changed to decimal using the pseudo-op DECIMAL (Section 5.11.10). Numbers are used in expressions.

5.8 SYMBOLS

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

5.8.1 Permanent Symbols

The assembler symbol table initially contains definitions of the symbols for all computer instructions and PAL8 pseudo-ops. These symbols are permanently defined by PAL8 and need no further definition by the user; they are summarized in Section 5.15. For example,

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

5.8.2 User-Defined Symbols

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

User symbols may be defined in two ways:

1. As a statement label. Labels are assigned a value equal to the current location counter.
2. As an explicitly-defined symbolic value. (For example, A = 33.)

Permanent symbols (instructions, special characters, and pseudo-ops) may not be redefined as a label or symbolic value. The following examples are legal labels:

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

The following are illegal labels:

AD>M,	(contains an illegal character)
7ABC,	(first character not alphabetic)
LA BEL,	(contains embedded spaces)
D+TAG,	(contains a legal but non-alphanumeric character)
LABEL,	(not immediately terminated by a comma)

5.8.3 Current Location Counter

As source statements are processed, PAL8 assigns consecutive memory addresses to the instructions and data words of the object program (binary and listing) being produced.

The current location counter contains the address in which the next word of object code will be assembled and is automatically incremented each time a memory location is assigned. A statement that generates a single object program storage word increments the location counter by one. Another statement might generate six storage words, incrementing the location counter by six.

The location counter is set or reset by typing an asterisk followed by an expression giving the address in which the next program word is to be stored. The expression may include symbols, but every such symbol must have been defined at some previous point in the current source file(s) being assembled. If the origin is not set by the user, PAL8 begins assigning addresses at location 200.

The symbol TAG in the following example is assigned a value of 0300, the symbol B a value of 0302, and the symbol A a value of 0303.

```
        *300            /SET CURRENT LOCATION COUNTER TO 300
TAG,    CLA
        JMP A
B,      0
A,      DCA B
```

If a symbol is defined more than once as a label, the assembler will display the "illegal definition" error message.

ID address

where address is the octal value of the location counter at the second occurrence of the symbol definition. The symbol is not redefined. PAL8 error conditions are described in Section 5.14. For example,

```
          *300
START,    TAD A
          DCA COUNTER
CONTIN,   JMS LEAVE
          JMP START
A,        -74
COUNTER,  0
START,    CLA CLL
          .
          .
          .
```

The symbol START would have a value of 0300; the symbol CONTIN would have a value of 0302; the symbol A would have a value of 0304; and the symbol COUNTER (considered COUNTE by the assembler, because the assembler uses only the first six characters of a symbol) would have a value of 0305. When the assembler processes the next line, it will display the error message:

ID COUNTE+0001

PAL8 will also display an error message if reference is made to an undefined symbol. For example,

```
        *7170
A,      TAD C
        CLA CMA
        HLT
        JMP A1
C,      0
```

This would produce the "undefined symbol" error message

US A+0003

since the symbol A1 has not been defined.

5.8.4 Symbol Table

Initially, the assembler's symbol table contains the definitions of the computer instructions and PAL8 pseudo-ops; these are PAL8's permanent symbols. As the source program is processed, user-defined symbols along with their 12-bit binary values are added to the symbol table. Entries in the symbol table are listed in alphabetic order at the end of the listing file produced, if any.

During pass 1, if PAL8 detects that the symbol table is full (in other words, there is no more memory space in which to store symbols and their associated values), the "symbol table exceeded" error message is displayed as follows:

```
SE address
```

and control returns to the Monitor. The number of symbols defined in the program may be reduced by using relative addressing for example, JMP START+3). It is also possible to segment a program and assemble the segments separately, taking care to define correct links between the segments. PAL8's symbol capacity is 2621 (decimal) symbols of which 96 are permanent. Where PAL8 is run under BATCH, 2357 symbols can be defined. (Use of the /K option expands these to 2971 and 2719 respectively, at the expense of slower assembly time.)

Instructions concerning altering the permanent symbol table are discussed in Section 5.11.8.

5.8.5 Direct Assignment Statements

New symbols with their assigned values may be inserted directly into the symbol table by using a direct assignment statement of the form

```
SYMBOL=VALUE
```

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

```
A=6
EXIT=JMP I 0
C=A+B
CO=JMS I [.]
```

All symbols to the right of the equal sign must already be defined, except that symbols are allowed to be undefined during pass 1. For example,

```
A=B
. . .
B=3
```

During pass 1, A will equal 0, since it is undefined thus far. During pass 2, A will equal 3, since B is given the value 3 at the end of pass 1. The use of the equal sign does not increment the location counter; it is an instruction to the assembler itself rather than a data value.

A direct assignment statement may also equate a new symbol to the value assigned to a previously defined symbol. For example,

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

The new symbol GAMMA is entered into the user's symbol table with the value 17. The value assigned to a symbol may be changed as follows:

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

The second line of code shown changes the value assigned to ALPHA from 5 to 7.

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

```
        *200
A=100           /DOES NOT UPDATE CURRENT LOCATION COUNTER
B=400           /DOES NOT UPDATE CURRENT LOCATION COUNTER
A+B             /THE VALUE 500 IS ASSEMBLED AT LOC 200
TAD A           /THE VALUE 1100 IS ASSEMBLED AT LOC 201
```

If the symbol to the left of the equal sign is in the permanent symbol table, the "redefinition" diagnostic

RD address

will be displayed as a warning, where address is the value of the location counter at the point of redefinition. The new value will be stored in the symbol table. For example,

CLA=7600

will cause the diagnostic

RD+200

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

5.8.6 Symbolic Instructions

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

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

5.8.7 Symbolic Operands

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

TOTAL, TAD ACI+TAG

The values of the two symbols ACI and TAG (previously defined in the program) are combined by a two's complement add. (See Section 5.9.1 on Operators.) This value is then used as the operand address.

5.9 EXPRESSIONS

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

5.9.1 Operators

Seven characters in PAL8 act as operators:

+	Two's complement addition
-	Two's complement subtraction
^	Multiplication (unsigned, 12-bit integer)
%	Division (unsigned, 12-bit integer)

!	Boolean inclusive OR
&	Boolean AND
Space (or TAB)	Treated as a Boolean inclusive OR except in a memory reference instruction

No checks for arithmetic overflow are made during assembly, and any overflow bits are lost from the high-order end. For example,

    7755+24

will give a result of 1.

The operators plus (+) and minus (−) may be used freely as unary (prefix) operators.

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

    3000^2

will give a result of 6000.

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

    7000%1000

will yield a result of 7. This example could be written as:

    -1000%1000

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

Use of the multiplication and division operators requires more attention to sign (on the part of the programmer) than is required for simple addition and subtraction. Table 5-2 contains examples of expressions using arithmetic operators.

Table 5-2 Use of Arithmetic Operators

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

The ! operator causes a Boolean inclusive OR to be performed bit by bit between the left-hand term and the right-hand term. Giving the /B option changes the memory of "!" throughout the assembly to become a 6-bit left shift of the left term prior to the inclusive OR of the right. According to this interpretation,

    If A=1 and B=2

then

    A!B=0102

Under normal conditions A!B would be 0003.

The & operator causes a Boolean AND to be performed bit by bit between the left and right values.

SPACE is an operator that has special significance depending on the context in which it is used. When the symbol preceding the space is not a memory reference instruction as in the following example

    SMA CLA

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

User-defined symbols are treated as non-memory reference instructions. For example,

```
A=1234
B=77
A B
```

stores a data value of 1277 (octal), the same as A!B.

If data values are generated, the current location counter is incremented. For example,

    B-7;A+4;A-B

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

    HLTCLA=HLT CLA

produces no information to be loaded (it produces a value for "HLTCLA" in the symbol table) and hence does not increment the current location counter.

In the program

```
            *4721
TEMP,
TEM2,       0
```

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

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

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

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

```
                *4100
                JMP A
A,              CLA
```

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

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

The seven address bits of A are taken, for example,

```
        000 001 000 001
```

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

```
        000 011 000 001
```

The operation code is then ORed into the JMP value to form

```
        101 011 000 001
```

or, in octal

```
        5301
```

In addition to the above tests, the page bits of the address field are compared with the page bits of the current location counter. If the page bits of the address field are nonzero and do not equal the page bits of the current location counter, an out-of-page reference is being attempted and the assembler will take action as described in Section 5.12 on Link Generation and Storage.

5.9.2 Special Characters

In addition to the operators described in the previous section, PAL8 recognizes several special characters that serve specific functions in the assembly process:

=	equal sign
,	comma
*	asterisk

.	dot
"	double quote
( )	parentheses
[ ]	square brackets
/	slash
;	semicolon
<>	angle brackets
$	dollar sign

The equal sign, comma, asterisk, slash, and semicolon have been previously described. The remaining special characters are described in the following sections.

5.9.2.1 **Dot (.)** – The special character dot (.) represents the current location counter. It may be used in any expression (except to the left of an equal sign), and must be separated from other symbols by a space or other operator. For example,

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

is equivalent to JMP 0202. Also,

```
*300
.+2400
```

will produce in location 0300 the quantity 2700. Consider

```
*140
PRINT=JMS I.
2200
```

The second line (PRINT=JMS I .) does not increment the current location counter; therefore, 2200 is placed in location 140 and PRINT is placed in the user's symbol table with an associated value of 4540 (the octal equivalent of JMS I .). This technique is useful in creating "global" subroutine calls.

Large buffers may be defined by using a format such as the following:

```
          *1200    /BUFFER LOCATION
BUFFER,   0        /FIRST WORD OF BUFFER
          *.+400   /DEFINE A 401 WORD BUFFER
NEXT,     0        /PROGRAM CONTINUES
```

5.9.2.2 **Double Quote (")** – When a double quote (") precedes an ASCII character, PAL8 assembles the 8-bit ASCII equivalent of the character. (ASCII codes are listed in Appendix A.) For example,

```
CLA
TAD       ("A)
```

The constant 0301 is placed in the accumulator when these two instructions are eventually executed. The character must not be a carriage return or one of the characters that is ignored on input (discussed at the end of this section).

5.9.2.3 **Parentheses ( ) and Brackets [ ]** – Left and right parentheses, ( ), enclose a current page literal (closing member is optional).

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

The left parenthesis is a signal to the assembler that the expression following is to be evaluated and assigned a word in the literal area of the current page. This is the same area in which the indirect address linkages are stored. In the above example, the quantity 2 is stored in a word in the literal area beginning at the end of the current memory page. The instruction in which the literal appears is given the address of the literal. A literal is assigned to storage the first time it is encountered; subsequent references to the same literal from the current page are made to the same location. The use of literals frees symbol storage table and makes programs much more readable. Literal allocation starts with the last location on the page and works towards the first location. If the literal area reaches the instruction/data area, a PE (Page Exceeded) error message is generated and assembly continues.

If square brackets ([and]) are used in place of parentheses, the literal is assigned to page zero rather than the current page. This enables a value to be referenced from any address within the field. For example,

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

The closing member is optional. Literals may contain any expression.

NOTE

Literals can be nested, for example:

```
*200
        TAD (TAD (30))
```

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

5.9.2.4 Angle Brackets (<>) – Angle brackets (<>) are used as conditional delimiters. The code enclosed in the angle brackets is to be assembled or ignored, depending on the definition of the symbol or value of the expression preceding the angle brackets. (The IFDEF, IFNDEF, IFZERO, and IFNZRO pseudo-operators are used with angle brackets and are described in Section 5.11.9.)

NOTE

Programs that use conditionals should avoid angle brackets in comments as they will be interpreted as beginning or terminating the conditional.

5.9.2.5 Dollar Sign ($) – The dollar sign ($) character is optional at the end of a program and is interpreted as an unconditional end-of-pass. It may however occur in a text string, comment, or double quote (") term, in which case it is interpreted in the same manner as any other character. This feature is provided for compatibility with older PDP-8 assemblers, and its use is not recommended.

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

FORM FEED	Used to skip to a new page
LINE FEED	Used to create a line spacing without causing a carriage return

Nonprinting characters include

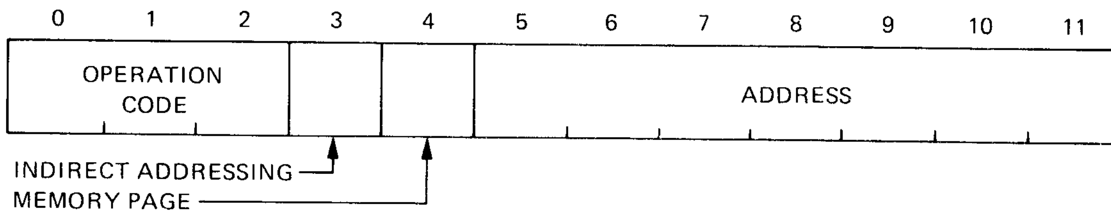
SPACE	Spaces to next character position
TAB	Spaces to next "tab column" of 8 characters
RETURN	Terminates each line
BELL (CTRL/G)	Sounds the terminal buzzer

5.10 INSTRUCTION SET

The instruction set for PAL8 includes processor instructions, input/output (I/O) instructions, and assembler instructions (pseudo-ops). The processor instructions are further divided into two basic groups of instructions: memory reference and microinstructions. (See Section 5.15 for detailed listing of instructions.)

5.10.1 Memory Reference Instructions

Memory reference instructions have the following format:



Memory Reference Bit Instructions

Bits 0 through 2 contain the operation code of the instruction to be performed. Bit 3 indicates whether the memory reference is indirect. Bit 4 indicates whether the instruction is referencing the current page rather than page zero. Bits 5 through 11 (7 bits) specify an address. Using these seven bits, 200 octal (128 decimal) locations can be directly specified; the page bit increases accessible locations to 400 octal or 256 decimal. A list of the memory reference instructions and their codes is given at the end of this chapter.

In PAL8, a memory reference instruction must be followed by one or more spaces and tabs, an optional I and/or Z designation, and any valid expression.

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

TAD 40

which is a direct address statement, where 40 is interpreted as the location on page zero containing the quantity to be added to the accumulator. References to locations on the current page and page zero may be done directly. For compatibility with older PDP-8 assemblers, the symbol Z is also accepted as a way of indicating a page zero reference, as follows:

TAD Z 40

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

TAD I 40

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

NOTE

Because the letter I is used to indicate indirect addressing, it may not be redefined as a label or variable. Likewise the letter Z, which is sometimes used to indicate a page zero reference, may not be redefined.

5.10.2 Microinstructions

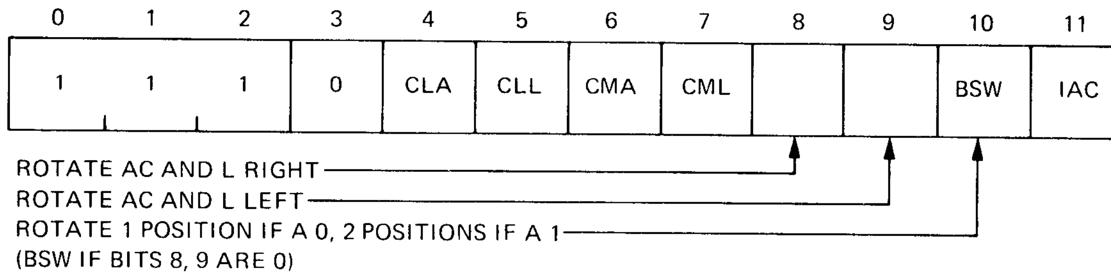
Microinstructions are divided into two classes: operate and Input/Output Transfer (IOT) microinstructions. Operate microinstructions are further subdivided into Group 1, Group 2, and Group 3 designations.

NOTE

If an illegal combination of microinstructions are specified, the assembler will perform an inclusive OR between them, resulting in an unexpected operation. For example,

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

5.10.2.1 Operate Microinstructions – Operate instructions are divided into three groups of micro instructions. Although it is possible to combine instructions within a group, it is not logically possible to combine instructions from different groups. Group 1 microinstructions perform clear, complement, rotate and increment operations on the Accumulator and Link registers, and are designated by the presence of a 0 in bit 3 of the machine instruction word.



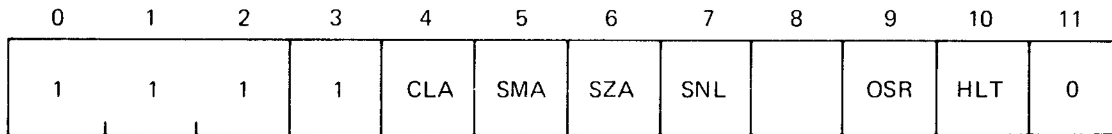
Group 1 Operate Microinstruction Bit Assignments

The following constants can be produced in the accumulator by a single instruction:

Constant	Instruction
0	CLA
1	CLA IAC
2	CLA CLL CML RTL
3*	CLA CLL CML IAC RAL
4*	CLA CLL IAC RTL
6*	CLA CLL CML IAC RTL
100*	CLA IAC BSW
2000	CLA CLL CML RTR
3777	CLA CLL CMA RAR
4000	CLA CLL CML RAR
5777	CLA CLL CMA RTR
6000*	CLA CLL CML IAC RTR
7775	CLA CLL CMA RTL
7776	CLA CLL CMA RAL
7777	STA (=CLA CMA)

Instructions that are starred (*) must not be used on software to be transported onto old (non-omnibus) PDP-8 computers.

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

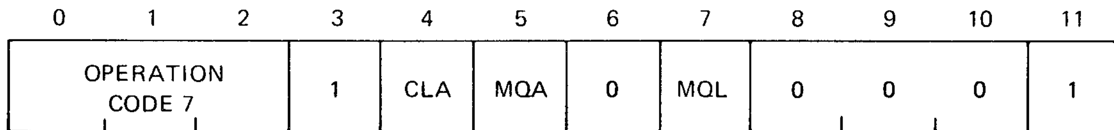


REVERSE SKIP SENSING OF BITS 5, 6, 7 IF SET

LOGICAL SEQUENCE: 1 (BIT 8 IS 0) - SMA OR SZA OR SNL
(BIT 8 IS 1) - SPA AND SNA AND SZL
2 - CLA
3 - OSR, HLT

Group 2 Operate Microinstruction Bit Assignments

Group 3 microinstructions reference the MQ register. They are differentiated from Group 2 instructions by the presence of a 1 in bits 3 and 11 of the machine instruction word.



CONTAINS A 1 TO SPECIFY GROUP 3
CONTAINS A 1 TO SPECIFY GROUP 3

Group 3 Operate Microinstruction Bit Assignments

Group 1 and Group 2 microinstructions cannot be combined since bit 3 determines either one or the other. Within Group 2, there are two groups of skip instructions. They can be referred to as the OR group and the AND group.

OR Group	**AND Group**
SMA	SPA
SZA	SNA
SNL	SZL

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

If skip instructions are combined, it is important to note the conditions under which a skip may occur.

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

   SZA SNL

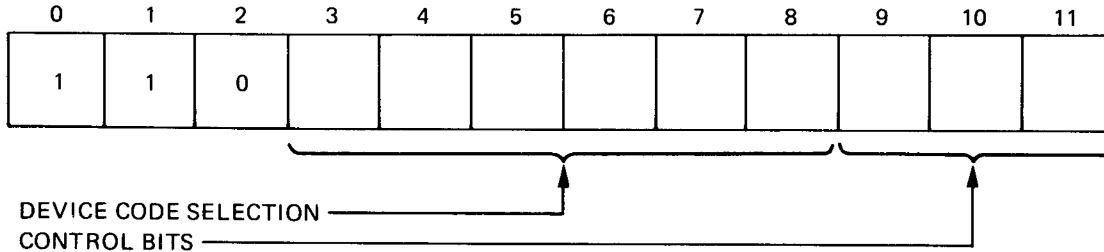
   The next statement is skipped if the Accumulator contains 0000 or the link is a 1 or both.

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

   SNA SZL

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

5.10.2.2 Input/Output Transfer Microinstructions – Input/output transfer microinstructions initiate operation of peripheral equipment and effect an information transfer between the central processor and the input/output device(s).



IOT Instruction Bit Assignments

5.10.3 Autoindexing

Consecutive address references are often necessary for obtaining data values when processing large amounts of data. Autoindex registers (locations 10-17 of each memory field) are used for this purpose. When one of the absolute locations from 10 through 17 (octal) is indirectly addressed, the contents of the location are incremented and then used as an indirect operand address. This allows consecutive memory locations to be adressed, using a minimum of instructions. It must be remembered that initially these locations (10 through 17 on page 0 of each field) must be set to one less than the first desired address. No incrementation takes place when locations 10 to 17 are addressed directly. For example, if the instruction to be executed next is in location 300 and the data to be referenced is on the the page starting at location 5000, autoindex register 10 can be used to address the data as follows:

```
0276     1377     TAD (4777)     /=5000-1
0277     3010     DCA 10         /SET UP AUTO INDEX
0300     1410     TAD I 10       /INCREMENT TO 5000
 .        .        .             /BEFORE USE OF AN INDIRECT
 .        .        .             /ADDRESS
 .        .        .
0377     4777     (Literal Area of Page 1)
```

Note that the Data Field must be set to the field of the data being referenced, in this case, Field 0.

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

5.11 PSEUDO-OPERATORS

Pseudo-operators are used to direct the assembler to perform certain processing operations 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. The pseudo-ops are defined in the permanent symbol table.

5.11.1 Indirect and Page Zero Addressing

The pseudo-operators I and Z specify the type of addressing to be performed. These were discussed in Section 5.10.1.

5.11.2 Extended Memory

The pseudo-op FIELD instructs the assembler to output a field setting so that it may assemble code into more than one memory field. This field setting is output during pass 2 in the object binary file and is recognized by the LOAD command which in turn causes all subsequent information to be loaded into the field specified by the expression. The form is

```
FIELD n
```

where n is an integer, a previously defined symbol, or an expression whose terms have been defined previously within the range 0 to 3.

This field setting is output to the binary file during pass 2 followed by an origin setting of 200. These settings are read by the LOAD command when it is executed to begin loading information into the new field.

The field setting is never remembered by the assembler except as the high-order digit of the Location Counter on the listing. A binary file produced without field settings will be loaded into field 0 when using the LOAD command.

When FIELD is used, the assembler follows the new FIELD setting with an origin at location 200. For this reason, to assemble code at location 400 in field 1, it would be necessary to write

```
FIELD:     /CORRECT EXAMPLE
*400
```

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

```
*400       /INCORRECT
FIELD:
```

Specifying the /0 option to PAL8 inhibits the origin to 200 after a FIELD pseudo-op, leaving the origin at its previous value.

5.11.3 Resetting the Location Counter

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

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

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

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

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

5.11.4 Reserving Memory

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

ZBLOCK n

For example,

ZBLOCK 40

causes the assembler to reserve 40 (octal) words and store zeros in them. The n may be an expression. If n=0, no locations are reserved. A ZBLOCK statement may have a label.

5.11.5 Relocation Pseudo-Operator

It is sometimes desirable to assemble code at a given location and then move it at run time to another location for execution. This may result in errors unless the relocated code is assembled in such a way that the assembler assigns symbols their execution-time addresses rather than their load-time addresses. The RELOC pseudo-op establishes a virtual location counter without altering the actual location counter. The line

RELOC expr

sets the virtual location counter to expr. The line

RELOC

resets the virtual location counter back to the actual location counter, terminating the relocation section.

For example, the following program

```
               0200                      *200
00200          1205                      TAD CHAR
00201          7402                      HLT
               1367                      RELOC 1367
01367*         1371                      TAD CODE
01370*         7402                      HLT
01371*         0000          CODE,       0
               0205                      RELOC
00205          0000          CHAR,       0
```

causes the assembler to load the ZERO word at CODE into location 204, but treats it as if it were loading into location 2000. The asterisks after the location values indicate that the virtual and the actual location counters differ for that line of code. Only the virtual location counter is listed. RELOC always causes current page literals to be dumped.

5.11.6 Suppressing the Listing

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

Two XLIST pseudo-ops may be used to enclose the code to be suppressed in which case the first XLIST with no argument will suppress the listing, and the second will allow it again. XLIST may also be used with an expression as an argument. The listing will be inhibited if the expression is not equal to zero, or allowed if the expression is equal to zero. XLIST pseudo-ops never appear in the assembly listing.

5.11.7 Controlling Page Format

The EJECT pseudo-op causes the listing to skip to the top of the next page. A page eject is done automatically every 55 lines; EJECT is useful if more frequent paging is desired. If this pseudo-op is followed by a string of characters, the first 40 (decimal) characters of that string will be used as a new title at the top of each page of the listing.

5.11.8 Altering the Permanent Symbol Table

If the system includes one or more optional devices that are to be programmed directly whose instruction set is not defined in the permanent symbol table (for example, line printer), then the symbol table would have to be altered to include instructions for these devices.

In another situation, programmed-defined symbols might require more space than available in the symbol table. Again, the symbol table would have to be altered by removing all definitions not needed in the program being assembled. For such purposes, PAL8 has three pseudo-ops that can be used to alter the permanent symbol table. These pseudo-ops are recognized by the assembler only during pass 1. During either pass 2 or pass 3, the assembler ignores them and they have no effect.

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

FIXTAB appends all currently defined symbols to the permanent symbol table. All symbols defined before the occurrence of FIXTAB are made part of the permanent symbol table for the current assembly.

To append the following instructions to the symbol table, it is necessary to generate an ASCII file called SYM.PA. This file contains the following:

```
CLSK=6131     /SKIP ON CLOCK INTERRUPT
FIXTAB        /SO THAT THIS WON'T BE
              /PRINTED IN THE SYMBOL TABLE
```

The ASCII file is then entered in the PAL8 input designation. The definitions could be placed at the beginning of the source file, to eliminate the need to load an extra file. Each time the assembler is loaded, the PAL8's initial permanent symbol table is restored.

The third pseudo-op used to alter the permanent symbol table in PAL8 is FIXMRI. FIXMRI defines a memory reference instruction and is of the form:

```
FIXMRI name=value
```

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

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

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

5.11.9 Conditional Assembly Pseudo-Operators

The IFDEF pseudo-op takes the form

```
IFDEF symbol <source code>
```

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

For example,

```
IFDEF A <TAD A
         DCA B>
```

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

```
IFNDEF symbol <source code>
```

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

The IFZERO pseudo-op is of the form

```
IFZERO expression <source code>
```

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

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

```
IFNZRO expression <source code>
```

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

Pseudo-ops can be nested. For example,

```
IFDEF SYM <IFNZRO X2 <...>>
```

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

Conditional code that is not assembled can be deleted from the listing output by using the /J option.

5.11.10 Radix Control

Numbers used in a source program are initially considered to be octal numbers. However, the program may change the radix interpretation by the use of the pseudo-operators DECIMAL and OCTAL. The DECIMAL pseudo-op interprets all following numbers as decimal until the occurrence of the pseudo-op OCTAL. The OCTAL pseudo-op resets the radix to octal.

5.11.11 Entering Text Strings

The TEXT pseudo-op allows a string of text characters to be entered as data and stored in 6-bit ASCII. The format required is the pseudo-op TEXT followed by one or more spaces or tabs, a delimiting character (must be a printing character), the string of text, and the same delimiting character. If the number of characters in a specified string is odd, the last word will contain zero in its right half. If the number of characters is even, a final word of zero will be appended. Either way, a final 6-bit character of zeros is generated providing a convenient "end-of-string" indication. Note that the /F option prevents the extra word of zero when the number of characters is even. Note also that six bits is only sufficient to encode the printing characters. For example:

```
TAG,    TEXT"123*"
```

The string would be stored as

```
6162
6352
0000
```

The /F option inhibits the generation of the extra 6-bit zero word. Alternatively, the statement "*.-1" may be used to eliminate the extra zero word (when the number of characters are even).

5.11.12 End-of-File Signal

PAUSE signals the assembler to stop processing the file being read. The current pass is not terminated, and processing continues with the next file. The PAUSE pseudo-op is present only for compatibility with paper tape assemblers, and its use is not recommended.

5.11.13 Use of DEVICE and FILENAME Pseudo-Operators

The pseudo-operators DEVICE and FILENAME may be used by calls to the User Service Routine (see Appendix C), or may be used for other purposes. They store 6-bit ASCII strings at the current location. The form for these pseudo-ops is

```
DEVICE name
FILENAME name.extension
```

The name used with DEVICE can be from 1 to 4 alphanumeric characters. These are trimmed to 6-bit ASCII and packed into two words, filled-in with zeros on the right if necessary. With FILENAME (FILENA is also acceptable), the name (or name.extension) may be from 1 to 6 alphanumeric characters and the optional extension may be 1 or 2 characters. The characters are trimmed to 6-bit ASCII and packed two to a word. Three words are allocated for the file name, filled with zeros on the right if fewer than 6 characters are specified, followed by one word for the extension.

For example,

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

is equivalent to the following coding:

```
L,  0102
    0300
    0000
    0401
```

The symbols DEVICE and FILENAME may not be used as labels since they are predefined pseudo-ops.

5.12 LINK GENERATION AND STORAGE

In addition to handling symbolic addressing on the current page of memory, PAL8 automatically generates "links" for off-page references. If a direct memory reference is made to an address not on the page where an instruction is located or on page 0, the assembler sets the indirect bit (bit 3) and an indirect address literal (a "link") will be stored on the current memory page. If the specified reference is already an indirect one, the error diagnostic II (Illegal Indirect) will be generated. In the following example,

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

the assembler will recognize that the register labeled A is not on the current page and will generate a link to it as follows:

1. In location 2600 the assembler will place the word 5777 (equivalent to JMP I 2777).
2. In address 2777 (the last available location on the current page) the assembler will place the word 2117 (the actual address of A).

In the listing, the octal code for the instruction will be followed by a single quote (') to indicate that a link was generated.

Although the assembler will recognize and generate an indirect address linkage when necessary, the program may indicate an explicit indirect address by the pseudo-op I. For example,

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

The assembler cannot generate a link for an instruction that is already specified as being an indirect reference, since the computer supports no such instruction format. For the example shown below, the assembler will print the error message II (Illegal Indirect).

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

NOTE

The option /E makes link generation a condition that produces the LG (Link Generated) error message.

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

Literals and links are stored on each page starting at page address 177 (relative) and extending toward page address 0 (relative). Whenever the origin is then set to another page, the literal area for the current page is output. There is room for 160 (octal) literals and links on page zero and 100 (octal) literals on each other page of memory. Literals and links are stored only as far down as the highest instruction on the page. Further attempts to define literals will result in a PE (Page Exceeded) or ZE (page Zero Exceeded) error message.

5.13 TERMINATING ASSEMBLY

PAL8 will terminate assembly and return to the Monitor under any of the following conditions:

1. Normal exit: The end of the source program was reached on pass 2 (or pass 3 if a listing is being generated).
2. Fatal error: One of the following error conditions was found and flagged (Section 5.14):

   BE DE DF PH SE

3. CTRL/C: If typed, control returns to the Monitor. Any partial output files are deleted.

5.14 ERROR MESSAGES

PAL8 will detect and flag error conditions and display error messages on the console terminal. The format of the error message is

```
COde address
```

where code is a 2-letter code that 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 label (if there was one) on the current page. For example, the code:

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

would produce the error message

```
IC BEG+0001
```

since % is an illegal character because of its placement.

In the listing, error messages are output as 2-character messages on the line just prior to the line in which the error occurred. Table 5-3 lists the PAL8 error codes. Fatal errors cause PAL8 to terminate the assembly immediately (deleting any output files produced so far) and return to the Monitor.

Table 5-3 PAL8 Error Codes

Error Code	Meaning
BE	A PAL8 internal table has overflowed. This situation can usually be corrected by decreasing the level of literal nesting or the number of current page literals used prior to this point on the page. Fatal error: assembly cannot continue.
CF	Chain to CREF error. CREF.SV was not found on SYS.
DE	Device error. An error was detected when trying to read or write a device. Fatal error: assembly cannot continue.

Continued on next page

Table 5-3 (Cont.) PAL8 Error Codes

Error Code	Meaning
DF	Device full. Fatal error: assembly cannot continue.
IC	Illegal character. The character is ignored and the assembly is continued.
ID	Illegal redefinition of a symbol. An attempt was made to give a previous symbol a new value by means other than the equal sign. The symbol is not redefined.
IE	Illegal equals. An attempt was made to equate a variable to an expression containing an undefined term. The variable remains undefined.
II	Illegal indirect. An off-page reference was made; a link could not be generated because the indirect bit was already specified.
IP	Illegal pseudo-op. A pseudo-op was used in the wrong context or with incorrect syntax.
IZ	Illegal page zero reference. The pseudo-op Z was found in an instruction which did not refer to page zero. The Z is ignored.
LD	The /L or /G options have been specified but the Absolute Loader system program is not present.
LG	Link generated. This code is displayed only if the /E option was specified to PAL8.
PE	Current nonzero page exceeded. An attempt was made to 1. Override a literal with an instruction. 2. Override an instruction with a literal. 3. Use more literals than the assembler allows on that page. This can be corrected by decreasing either the number of literals on the page or the number of instructions on the page.
PH	A conditional assembly bracket is still in effect at the end of the input stream. This is caused by nonmatching < and > characters in the source file.
RD	Redefinition. A permanent symbol has been defined with =. The new and old definitions do not match. The redefinition is allowed.
SE	Symbol table exceeded. Too many symbols have been defined for the amount of memory available. Fatal error: assembly cannot continue.
UO	Undefined origin. An undefined symbol has occurred in an origin statement.
US	Undefined symbol. A symbol has been processed during pass 2 that was not defined before the end of pass 1.
ZE	Page 0 exceeded. This is the same as PE except occurs in page 0.

5.15 PAL8 PERMANENT SYMBOL TABLE

The following mnemonics represent the central processor's instruction set found in the permanent symbol table within the PAL8 Assembler. For additional information on these instructions, refer to the *DECstation 78 User's Guide.*

Mnemonic	Code	Operation
Memory Reference Instructions		
AND	0000	Logical AND
TAD	1000	Two's complement add
ISZ	2000	Increment and skip if zero
DCA	3000	Deposit and clear AC
JMS	4000	Jump to subroutine
JMP	5000	Jump
IOT	6000	In/Out transfer
OPR	7000	Operate
Group 1 Operate Microinstructions		
NOP	7000	No operation
IAC	7001	Increment AC
BSW	7002	Byte swap
RAL	7004	Rotate AC and Link left one
RTL	7006	Rotate AC and Link left two
RAR	7010	Rotate AC and Link right one
RTR	7012	Rotate AC and Link right two
CML	7020	Complement the link
CMA	7040	Complement the AC
CLL	7100	Clear Link
CLA	7200	Clear AC
Group 2 Operate Microinstructions (1 cycle)		
HLT	7482	Halts the computer
SKP	7410	Skip unconditionally
SNL	7420	Skip on nonzero Link
SZL	7430	Skip on zero Link
SZA	7440	Skip on zero AC
SNA	7450	Skip on nonzero AC
SMA	7500	Skip on minus AC
SPA	7510	Skip on positive AC (zero is positive)
Group 3 Operate Microinstructions		
MQA	7501	MQ OR into AC
MQL	7421	Load MQ, clear AC
SWP	7521	Swap AC and MQ
Combined Operate Microinstructions		
CIA	7041	Complement and increment AC
STL	7120	Set Link to 1
GLK	7204	Get Link (put Link in AC, bit 11)
STA	7240	Set AC to -1 (7777)

Mnemonic	Code	Operation
Internal IOT Microinstructions		
SKON	6000	Skip with interrupts on and turn them off
ION	6001	Turn interrupt facility on
IOF	6002	Turn interrupt facility off
GTF	6004	Get flags
RTF	6005	Restore flag, ION
CAF	6007	Clear all flags
Memory Extension IOT Instructions		
CDF	62N1	Change the Data Field to N
CIF	62N2	Change Instruction Field to N at the next JMP or JMS instruction
CDF CIF	62N3	Combined CDF and CIF
RDF	6214	Read (OR) the Data Field into bits 6-8 of AC
RIF	6224	Read (OR) the Instruction Field into bits 6-8 of AC
RSB	6234	Read Instruction Save Field into bits 7-8 and Data Save Field into bits 10-11 of AC
RMF	6244	Restore memory fields to state prior to last interrupt by loading the Data Save Field into DF register and the Instruction Save Field into the IB register and inhibiting interrupts. At the next JMP or JMS, IB is loaded into the IF register and the interrupt inhibit is removed.
Keyboard (1 cycle)		
KCF	6030	Clear keyboard flag
KSF	6031	Skip on keyboard flag
KCC	6032	Clear keyboard flag and AC
KRS	6034	OR keyboard buffer into AC
KIE	6035	Set/clear interrupt enable
KRB	6036	Clear AC, read keyboard buffer Clear keyboard flag
Terminal (1 cycle)		
TSF	6041	Skip on terminal flag
TCF	6042	Clear terminal flag
TSK	6045	Skip on keyboard or terminal flag
TLS	6046	Load terminal, display character, and clear terminal flag

CHAPTER 6
BASIC

6.1 INTRODUCTION

OS/78 BASIC* is an interactive programming language used in scientific and business environments to solve mathematical problems with a minimum of programming effort. It also is used by educators and students as a problem-solving tool and as an aid to learning through programmed instruction and simulation.

In many respects the BASIC language is similar to other programming languages (such as FORTRAN IV), but BASIC is aimed at facilitating communication between the user and the computer. BASIC simply requires that you type in the computational procedure as a series of numbered statements, making use of common English words and familiar mathematical notations. Because of the small number of commands necessary and its easy application in solving problems, BASIC is an easy computer language to learn. With experience, the advanced techniques available can be added in the language to perform more intricate manipulations or to express a problem more efficiently and concisely.

6.2 MAJOR COMPONENTS OF OS/78 BASIC

The BASIC subsystem has four major components.

1. BASIC Editor (BASIC.SV)
2. Compiler (BCOMP.SV)
3. Loader (BLOAD.SV)
4. BASIC Run Time System (BRTS.SV, BASIC.AF, BASIC.SF, BASIC.FF)

The BASIC editor is used to create and edit program source files. During this process, the editor creates a file called BASIC.WS containing the current program.

Once the program has been prepared for execution, entering a RUN command causes the editor to chain to the BASIC compiler. The compiler converts the statements in BASIC.WS into relocatable binary instructions.

Following compilation, the BASIC loader is automatically requested. The loader converts the relocatable binary data output by the compiler into executable form and loads the result into memory.

The BASIC loader then chains to the BASIC Run Time System (BRTS) which executes the program. The modules BASIC.AF, BASIC.FF, and BASIC.SF are overlays to BRTS.SV.

6.3 BASIC INSTRUCTION REPERTOIRE

BASIC instructions and commands can be grouped in three categories as follows:

1. BASIC Editor commands that allow you to

   a. Create or modify a program,
   b. Execute a program,
   c. Retrieve a program from diskette, and
   d. Save a program.

---

*BASIC is a registered trademark of the trustees of Dartmouth College.

2. BASIC statements, comprising the BASIC language, that are the building blocks used to create and structure BASIC programs.
3. BASIC Functions, represented by subroutines, that are built into BASIC primarily to facilitate problem solving activities.

6.4 CALLING BASIC

To enter the BASIC subsystem, type

```
.BASIC
```

in response to the Monitor dot. This command invokes the BASIC editor.

6.5 BASIC EDITOR COMMANDS

6.5.1 Using the BASIC Editor

The BASIC editor incorporates all the tools and capabilities necessary to create, correct, modify, execute, save and retrieve BASIC programs. After calling BASIC, the BASIC editor responds with the displayed query

```
NEW OR OLD--
```

Editing is now continued in one of two ways:

1. Typing NEW with a file name instructs the system to initiate the creation of a new file.
2. Typing OLD with a file name instructs the system to retrieve an old file containing a previously-generated program.

Figure 6-1 summarizes user actions implemented by the BASIC editor. This figure shows two arbitrarily selected file names (MAKER/ALTER). File names may contain no more than six alphanumeric symbols.

The left side of Figure 6-1 shows the types and patterns of activities that you ordinarily pursue after typing the NEW command. The right side of the illustration shows activities frequently carried out after typing the OLD command. Note that most of the BASIC editor commands appear on both sides of the figure. Only the sequence in which they are used differs.

6.5.2 BASIC Editor Commands

This section summarizes the BASIC editor commands. Letters that are not required in naming the command are shown as lower case in the Command/Parameter description. For example, only NE is required to be recognized by the BASIC editor as the NEW command.

NOTE

The RETURN key must be pressed following each BASIC editor command.

6.5.2.1 NEW Command – The NEW command clears the memory workspace and specifies the name of the program that is to be input.

Command	**Parameters**
NEw	file [.ex]
file.ex	is the new file name and extension of the program about to be typed in. If the extension is omitted, the editor assigns .BA.

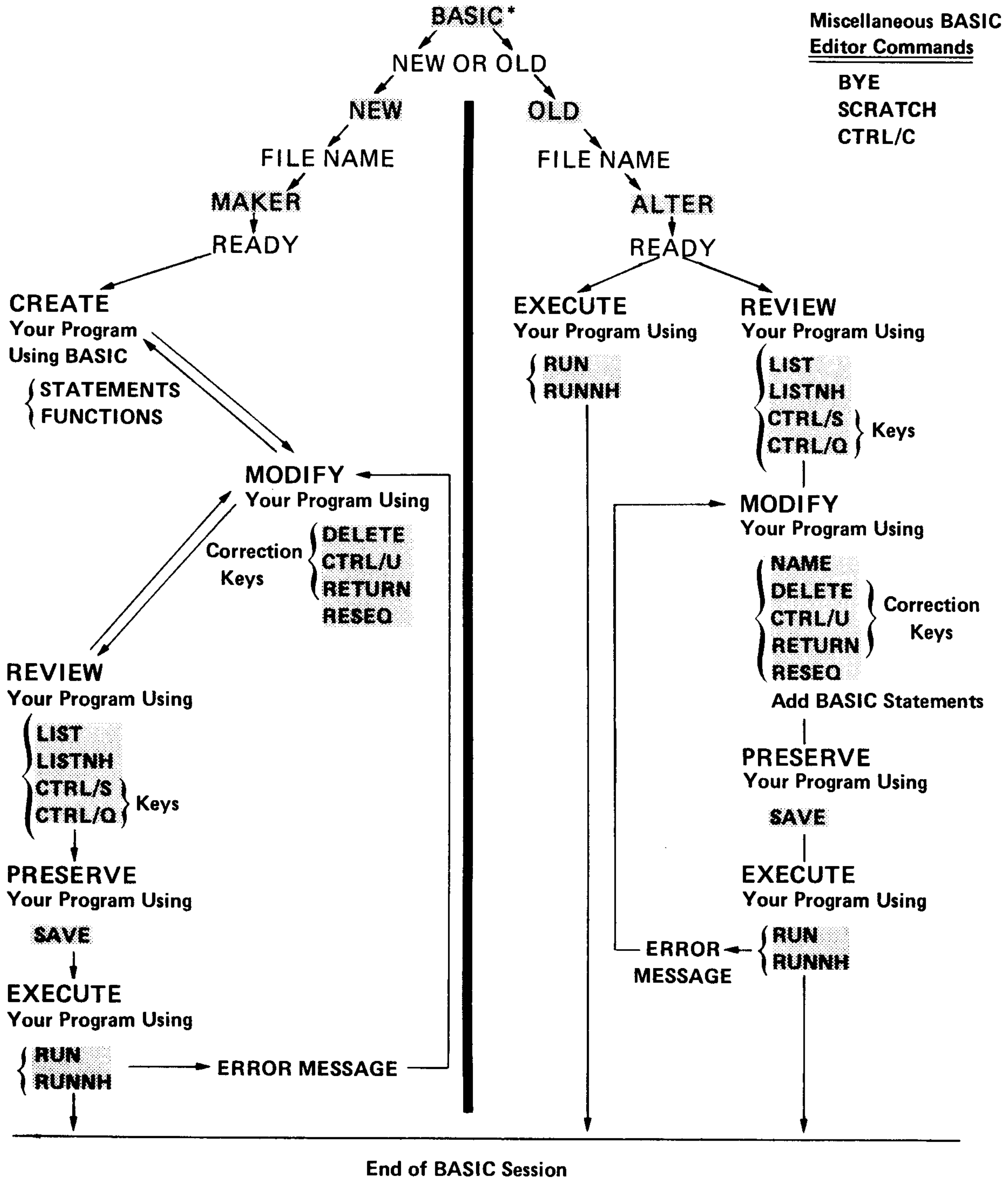


Figure 6-1 BASIC Editor Commands and Related Uses

An alternate method is to type NEW without a file name, followed by the RETURN key. BASIC displays

```
FILE NAME --
```

in response to which the file name and extension is typed.

For example, to clear the workspace and name the new program "TEST.BA", type

```
NEW TEST
```

or

```
NE TEST.BA
```

6.5.2.2 **OLD Command** – The OLD command clears the memory workspace, and causes the editor to find a program on a diskette and place it into the workspace.

Command	**Parameters**
OLd	dev:file[.ex]
dev:file.ex	is the device, file name, and extension of the program on the disk. If the extension is omitted, BASIC assumes ".BA".

Another method is to type OLD without a file name. BASIC displays

```
FILE NAME --
```

in response to which the device, file name, and extension is typed. When no device is specified, the BASIC editor defaults to DSK (usually = SYS).

For example, to bring TEST.BA into the workspace from RXA1, type

```
OLD RXA1:TEST.BA
```

or

```
OL RXA1:TEST
```

6.5.2.3 **LIST/LISTNH Commands** – The LIST command displays the current program along with a header line, containing the program name, date, and the revision number of BASIC. The date is displayed only if the current date has been entered into the system.

Command	**Parameters**
LIst	[n]

If n is omitted all program statements in the workspace are displayed. When n is specified, line n and all subsequent lines are displayed. Type CTRL/O to terminate a listing.

For example, typing LIST (or LI) displays the program PROG.BA:

```
LIST

PROG BA       5A       26-JUL-77

10 FOR A=1 TO 5
20 PRINT A
30  NEXT A
40  END

READY
```

Typing LI 30 displays line 30 and all subsequent lines:

```
LI 30

PROG BA       5A       26-JUL-77

30 NEXT A
40 END

READY
```

The LISTNH command also displays the program statements in the workspace but without the header.

Command	Parameters
LISTNH	[n]
n	has the same effects as specified for the LIST command, but does not display the header.

6.5.2.4 SAVE Command – The SAVE command writes the program in the workspace onto the diskette as a permanently saved file. Do not confuse this command with the Monitor SAVE command.

Command	Parameters
SAve	[dev:file.ex]
dev	is the device on which you want to store your program.
file.ex	is the file name and extension that the program will have on the diskette. If both are left out, BASIC will use the current file name and extension of the program in the workspace. If only the extension is omitted BASIC assigns .BA. If DEV: is omitted, BASIC assumes the device is DSK.

In the following example, the program "TEST.BA" is in the workspace. To store it on RXA1 under the same file name and extension, type

```
SAVE RXA1:TEST.BA
```

or

```
SA RXA1:TEST
```

6.5.2.5 RUN/RUNNH Commands – The RUN command executes the program in the workspace. Note that this command differs from the monitor RUN command.

Command	Parameters
RUn	(none)

Prior to program execution, this command displays a heading consisting of the file name and extension, BASIC version number and system date, assuming that the current date has been entered in the system. When program execution is completed, BASIC displays

READY

The following example shows the program PROG.BA in the workspace displaying the result of a calculation:

```
RUN
PROG BA        5A        26-JAN-76
3.179
```

The RUNNH command executes the program in the workspace, but does not display the header line.

Command	Parameters
RUNNH	(none)

Thus, the only difference between the RUNNH and RUN commands is that RUN prints the header and RUNNH does not display the header.

6.5.2.6 NAME Command – The NAME command allows the user to rename the program in the workspace.

Command	Parameters
NAme	newfil[.ex]
newfil.ex	is the new file name and extension of the program in the workspace. If the extension is omitted, the editor assigns .BA.

To change the name of the program in the workspace to PROG.BA, type

```
NAME PROG.BA
```

or

```
NA PROG
```

6.5.2.7 SCRATCH Command – The SCRATCH command erases all statements from the workspace, that is, it clears the workspace.

Command	Parameters
SCratch	(none)

6.5.2.8 BYE Command – The BYE command exits from BASIC and returns control to the Monitor from BASIC.

Command	Parameters
BYE	(none)

Typing BYE before SAVEing a newly created program will delete the program. Whenever the BASIC editor is waiting for user input, typing CTRL/C performs the same function as BYE.

6.5.3 BASIC Control Keys

This section describes the control keys that are used to correct errors, eliminate and substitute program links, and control program listings.

6.5.3.1 Correcting Typing and Format Errors (DELETE, CTRL/U) – Errors made while typing programs at the terminal are easily corrected. Pressing the DELETE key causes deletion of the last character typed. One character is deleted each time the key is pressed.

Sometimes it is easier to delete a line being typed and retype the line rather than attempt a correction using a series of DELETEs. Typing CTRL/U will delete the entire line currently being worked on and echoe "DELETED" and a carriage return-line feed. Use of the CTRL/U key is equivalent to typing DELETEs back to the beginning of the line.

6.5.3.2 Eliminating and Substituting Program Lines (RETURN) – To delete a program line that has already been entered into the computer, simply type the line number and then press the RETURN key. Both the line number and the statement(s) are removed from the program.

Change individual lines by simply retyping them in again. Whenever a line is entered, it replaces any existing line having the same line number. New lines may be inserted anywhere in the program by giving them unique line numbers.

6.5.3.3 Interrupting Program Execution – Program execution may be terminated by typing CTRL/C. BASIC responds by displaying the READY message allowing you to correct or add statements to the program.

NOTE

BASIC responds to CTRL/C with a "READY" message only if you have already given the RUN command. Typing CTRL/C when the BASIC Editor is operational causes a return to the Monitor.

6.5.3.4 Controlling Program Listings at the Terminal Console (CTRL/S, CTRL/C and CTRL/O) – For programs exceeding a single display frame (24 lines) the user may wish to stop the scrolling effect that occurs after typing the LIST/LISTNH command. Three sets of control keys are provided to do this. They are as follows:

1. CTRL/S keys. Simultaneously pressing these keys suspends listing (scrolling) of the program. However it leaves the program in a state where you may resume listing.
2. CTRL/Q keys. Simultaneously pressing these keys (after having suspended scrolling with CTRL/S), resumes the listing process.
3. CTRL/O keys. Simultaneously pressing these keys aborts the listing process and causes the BASIC editor to display READY.

6.5.4 Resequencing Programs (RESEQ)

If a program is extensively modified, you may find that some portions of the program have line numbers spaced so closely together that they do not permit any further addition of statements. Renumbering the lines in the program to provide a practical increment between line numbers can be done by using the RESEQ program. Note that the RESEQ program modifies the line numbers in GOSUB and IF-THEN statements to agree with the new line numbers assigned to program statements by RESEQ. Line lengths must not exceed 80 characters and programs may not exceed 350 lines.

Typically, the program would be used as follows:

SAVE DSK:SAMPLE BA	User saves program SAMPLE which requires renumbering.
READY	BASIC is ready for next command.
OLD DSK:RESEQ	User calls for program RESEQ.
READY	BASIC is ready for next command.
RUNNH	User runs RESEQ program.
FILE:DSK:SAMPLE.BA	Program asks for filename. User responds with device, name, and extension of program to be renumbered.
START,STEP:100,10	Program asks for a starting line number (START) and for the increment between line numbers (STEP). User requests that SAMPLE start with line number 100 and each line be incremented by 10.
READY	Renumbering is accomplished. BASIC ready for next command.
OLD DSK:SAMPLE.BA	User calls back his program.
READY	BASIC ready for next command.
LISTNH	User gets listing of program SAMPLE for further modification.

6.6 DATA FORMATS ACCEPTABLE TO BASIC

6.6.1 Numeric Information

6.6.2 Numbers

Numbers are expressed in decimal or E (exponential) format. Examples of numbers in both categories are as follows:

Decimal	E Type
0	10.23E27
7	6.21E+27
+69	-7.232E6
-52	2.211E-3
-3.9265	-2.2114E-4
0.123	
-0.769	

In the decimal format, the decimal point is optional for integers. That is, BASIC assumes a decimal point after the rightmost digit of an integer. In E type format, BASIC assumes a positive exponent when no plus (+) or minus (-) sign follows the E. Substitute the words "times ten to the power of" for the letter E when reading E type format numbers.

Numeric data may be input in either format. Results of computations with an absolute value outside the range +.000001<N<999999 are always output in E type format. BASIC handles six significant digits as shown by the following examples:

Value Typed In	Value Output by BASIC
.01	0.0099999
.0099	0.0099
999999	999999
100000	.100000E+007
.0000009	.899999E-006

Note in the above examples that the nature of the binary numbering system does not permit a completely accurate representation of certain decimal numbers. Hence they are output as close to the true value as the internal logic of the computer permits.

BASIC automatically suppresses the printing of leading and trailing zeros in integer numbers and all but one leading zero in decimal numbers. As can be seen from the preceding examples, BASIC formats all exponential numbers in the form:

sign .xxxxxxE(+or-)n

where x represents the number carried to six decimal places, E stands for "times 10 to the power of", and n represents the exponential value.

For example,

-.347021E+009 is equal to -347,021,000, and

.726000E-003 is equal to 0.000726

All numbers used in BASIC must have an absolute value (N) in the range:

10^-616<N<10^+616

6.6.3 Simple Variables

A simple variable in BASIC is an algebraic symbol representing a number, and is formed by a single letter or a letter followed by a digit. For example,

Acceptable Variables	Unacceptable Variables
J	2C (a digit cannot begin a variable)
B3	AB (two or more letters cannot form a variable)

Values may be assigned to variables either by indicating the values in LET statements, or by inputting the values as data via INPUT and DATA statements.

Examples:

```
10 LET I=53721
20 LET B3=456.9
30 LET X=20E9
40 INPUT Q
50 DATA 5,6,7
```

6.6.4 Subscripted Variables
In addition to simple variables, BASIC accepts another class of variables called subscripted variables. Subscripted variables provide additional computing capabilities for handling arrays, lists, tables, matrices, or any set of related variables. Variables are allowed one or two subscripts. A single letter or a letter followed by a digit forms the name of the variable. The subscript is formed by one or two integers enclosed in parentheses and separated by commas. Up to 31 arrays are possible in any program, subject only to the amount of memory available for data storage. For example, an array might be described as A(J) where J goes from 1 to 5, as follows:

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

This allows reference to be made to each of the five elements in the array A. A two-dimensional array A(J,K) can be defined in a similar manner, but the subscripted variable A must always have the same number of subscripts (that is, A(J) and A(J,K) cannot be used in the same program). It is possible, however, to use the same variable name as both a subscripted and an unsubscripted variable. Both A and A(J) are valid variable names and can be used in the same program. For more information on arrays and the use of subscripts, see the description of the DIM statement (Section 6.7.10.1).

6.6.5 Arithmetic Operations

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

Symbol Operator	Meaning	Example
+	Addition	A+B
–	Subtraction	A–B
*	Multiplication	A*B
/	Division	A/B
^(or**)	Exponentiation (Raise A to the B Power)	A^B or (A**B)

6.6.5.2 Priority – In any given mathematical formula, BASIC performs the arithmetic operations in the following order:

1. Parentheses receive top priority. Any expression within parentheses is evaluated before an unparenthesized expression.
2. In the absence of parentheses, the order or priority is:

   a. Exponentiation
   b. Multiplication and Division (of equal priority)
   c. Addition and Subtraction (of equal priority)

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

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

1. A+B = step 1
2. (result of step 1)-C = answer

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

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

6.6.5.3 Parentheses – Parentheses may be used to change the order or priority because expressions within parenthesis are always evaluated first. Thus, by enclosing expressions appropriately, the order of evaluation can be controlled. Parentheses may be nested, that is, enclosed by one or more sets of parentheses. In this case, the expression within the innermost parentheses is evaluated first, and then the next innermost, and so on, until all have been evaluated. Consider the following example:

```
A=7*((B^2+4)/X)
```

The order of priority is:

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

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

```
A*B^2/7+B/C+D^2
((A*B^2)/7)+((B/C)+D^2)
```

Both of these formulas will be executed in the same way. However, the second example may be easier to understand. Spaces may also be used to increase readability. Since the BASIC compiler ignores spaces, the two statements:

```
10 LET B=D^2+1
10LETB=D^2+1
```

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

6.6.5.4 Rules for Exponentiation – The following rules apply in evaluating the expression A^B.

Rule	Example
1. If B=0, then A^B=1	3^0=1
2. If A=0 and B>0, then A^B=0	0^2=0
3. If A=0 and B<0, then A^B=0 and a DV error message is displayed	0^-2=0
4. If B is an integer >9, then A^B= $A_1 * A_2 * A_3 \ldots * A_n$, where n=B	3^5=3*3*3*3*3=243
5. If B is an integer <0 then A^B= $1/A_1 * A_2 * A_3 \ldots * A_n)$, where n=B	3^-5=1/243
6. If B is a decimal (non-integer) and A>0, then A^B=EXP(B*LOG(A))	2^3.6= $e^{B1nA} = e^{3.61n2}$
7. If B is a positive or negative decimal (non-integer) and A>0, the program halts and an EM error is displayed.	-3^2.6 is illegal.

6.6.5.5 Relational Operators – A program may require that two values be compared at some point to discover their relation to one another. To accomplish this, BASIC makes use of the following relational operators:

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

Depending upon the result of the comparison, control of program execution may be directed to another part of the program. Relational operators are used in conjunction with the IF-THEN statement.

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

6.6.6 String Information

The previous sections dealt only with numerical information. However, BASIC also processes alphanumerical information called strings. A string is a sequence of characters, each of which is a letter, a digit, a space, or some character other than a statement terminator (backslash or carriage return).

6.6.6.1 String Character Set – The character set recognized by BASIC is as shown below. The decimal code for each character is also shown.

Code Number		Code Number	
0	@	18	R
1	A	19	S
2	B	20	T
3	C	21	U
4	D	22	V
5	E	23	W
6	F	24	X
7	G	25	Y
8	H	26	Z
9	I	27	[(left bracket)
10	J	28	\ (back slash)
11	K	29	] (right bracket)
12	L	30	^ (exponent sign)
13	M	31	_ (underscore)
14	N	32	(space)
15	O	33	! (exclamation point)
16	P	34	" (double quotes)
17	Q	35	#

Code Number		Code Number	
36 $		50 2	
37 %		51 3	
38 &	(ampersand)	52 4	
39 '	(apostrophe)	53 5	
40 (		54 6	
41)		55 7	
42 *		56 8	
43 +		57 9	
44 ,	(comma)	58 :	
45 -	(hyphen or minus sign)	59 ;	
46 .	(period)	60 <	(left-angle bracket)
47 /	(slash or division sign)	61 =	
48 0		62 >	(right-angle bracket)
49 1		63 ?	

6.6.6.2 String Conventions – Strings may be used as constants or variables. String constants are enclosed in quotes. For example,

"THIS IS A STRING CONSTANT"

A string variable consists of a letter followed by a dollar sign ($) or a letter and a single digit followed by a dollar sign. A$ and A1$ are both legitimate string variable names while 2A$ and AA$ are not.

A string variable may contain at most eight characters unless it has been dimensioned with the DIM statement. Quotation marks may be included in strings by indicating two quotation marks in succession. For example, the string A"B would appear in a program as:

```
10 LET A$="A""B"
```

The following lines:

```
10 LET A$="""QUOTE"""
20 PRINT A$
99 END
```

will result in this display:

```
"QUOTE"
```

It is important to recognize the real, structural difference between strings and numerical data. This number

2

is not identical to this string:

"2"

Numerical data may not be used where strings are required, and vice-versa.

6.6.6.3 String Concatenation – Strings can be concatenated, that is, connected like links in a chain, by using the ampersand (&). For example, as a result of the following lines, the next statement executed will be line 460:

```
400 LET A1$ = "AB"
410 LET B1$ = "CD"
420 LET C1$ = "ABCD"
430 IF C1$=A1$&B1$ GOTO 460
```

The ampersand can be used to concatenate string expressions wherever a string expression is legal, with the exception that information cannot be stored by means of a LET statement in concatenated string variables. That is, concatenated string variables cannot appear to the left of the equal sign in a LET statement. For example, this statement is legal

```
LET A$=B$&C$
```

while this statement is not:

```
LET A$&B$=C$
```

6.6.7 Format Control Characters

In OS/78 BASIC, a terminal line is formatted into five fixed zones (called print zones) of 14 columns each. A program such as:

```
1 LET A=2.3
2 LET B=21
3 LET C=156.75
4 LET D=1.134
5 LET E=23.4
10 PRINT A,B,C,D,E
15 END
```

where the control character comma (,) is used to separate the variables in the PRINT statement will cause the values of the variables to be displayed, using all five zones. For example,

```
RUNNH

2.3           21            156.75        1.134         23.4

READY
```

It is not necessary to use the standard five zone format for output. The control character semicolon (;) causes the text or data to be output immediately after the last character printed.

The following example program illustrates the use of the control characters in PRINT statements.

```
5  READ A,B,C,D,E,F
10 PRINT A,B,C,D,E,F
15 PRINT
20 PRINT A;B;C;D;E;F
25 DATA 4,5,6
30 DATA 16,25,36
```

```
RUNNH
4               5               6               16              25
36

4 5 6 16 25 36

READY
```

As this example illustrates, when more than five variables are listed in the PRINT statement, OS/78 BASIC automatically moves the sixth number to the beginning of the next line.

6.6.8 Files

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

A fixed length file is one which is already in existence. That is, it has been created and CLOSEd. The length of a fixed length file is equal to the number of blocks in the file and cannot be changed.

A variable length file is a newly created file. Until the file is CLOSEd, it is equal in length to the largest free space on the device. When the file is CLOSEd it becomes a fixed length file equal in length to the actual number of blocks it occupies. Unless the file is CLOSEd, the CHAIN, STOP or END statements will cause a loss of the file.

6.7 BASIC STATEMENTS

BASIC statements are the principal components of BASIC programs. The general format of BASIC statements is:

    xxx   Word Type Parameters

where

xxx is the line number;

Word Type is the statement (instruction) type; and

Parameters are the variables used in conjunction with the statement type.

Each statement starts with a line number followed by the word statement type. Spaces have no significance in BASIC language statements except in messages or literal strings which are displayed or printed out. Thus, spaces may, but need not, be used to modify a program and make it more readable.

Multiple statements may be placed on a single line by separating each statement from the preceding statement with a backslash. For example:

```
10 LET A=5\LETB=.2\LETC=3\PRINT "ENTER DATA"
```

All of the statements in line 10 will be executed before BASIC continues to the next line. Only one statement number at the beginning of th entire line is necessary. However, it should be remembered that program control cannot be transferred to a statement within a line, only to the first statement of the line in which it is contained. This consideration is important when using control statements or loop statements (see related descriptions).

6.7.1 Statement Line Numbers (Sequencing)

Failure to assign a line number results in the message:

```
WHAT
```

Each line of the program begins with a line number of 1 to 5 digits that serves to identify the line as a statement. The largest allowable line number is 99999.

A common programming practice is to number lines by fives or tens, so that additional lines may be inserted in a program without the necessity of renumbering lines already present. Renumbering a program can be accomplished by using the RESEQ program (Section 6.5.4).

6.7.2 The PRINT Statement

The PRINT statement is used to perform calculations and display results. It is also used to display alphanumeric (string) messages.

PRINT Statement Form

Line Number	Statement	Parameter
xxx	PRINT	expressions

Where expressions may be numbers, variables, strings or arithmetic expressions, separated by commas or semicolons. When used without an expression, a blank line will be output on the terminal.

In BASIC, a line is formatted into five fixed zones (called print zones) of 14 columns each. If the expressions in a PRINT statement are separated by commas, each will begin at a 14-column interval. That is, the first expression will be displayed starting at the first position, the second at the fifteenth position, and so forth. If more than five variables are involved, the display will automatically continue at the beginning of the next line.

If this format is not desired, you may separate expressions with the semicolon (;), causing the text or data to be output immediately after the last character printed (see Example 3).

If the last expression in a PRINT statement is followed by a comma or semicolon, the next display called for by the program will begin on the same line (if there is room). This also applies to the question mark displayed by the INPUT statement (see Examples 2 and 4).

Any algebraic expression in a PRINT statement will be evaluated using the current value of the variables.

Regardless of format (integer, decimal, or E-type), BASIC prints numbers in the form:

sign number space

where the sign is either minus (-) or blank (for plus) and a blank space always follows the number (see Example 3).

Strings and numeric expressions may be combined in a single PRINT statement (see Example 3).

To output an alphanumeric message, enclose the expression in quotation marks. To print a quotation mark (") put two quotation marks ("") in the expression (see Example 5).

The following examples illustrate the use of the PRINT statement.

Example 1:

The following lines

```
40 LET A=1
50 LET B=2
55 LET C=3
60 PRINT A,B,C
99 END
```

will cause each variable to begin at a 14-column interval as follows:

```
1             2             3
```

Example 2:

The following lines

```
110 LET A$="PRINTING"
120 LET B$= "STRINGS"
130 PRINT A$,
140 PRINT B$
199 END
```

will display the following:

```
PRINTING      STRINGS
```

Example 3:

The following lines:

```
10 A=5
20 PRINT "NUMBER";A;"AND";6
99 END
```

will cause this display:

```
NUMBER 5 AND 6
```

Blanks appear before the 5 and 6 because they are positive. The blank following the 5 accounts for the blank that BASIC always generates after a number.

Example 4:

The following lines

```
60 PRINT "NUMBER OF YEARS";
70 INPUT N
99 END
```

will display (the number 9 is typed in by the user) the following:

```
NUMBER OF YEARS
?9
```

Example 5:

The following lines

```
80 PRINT "MESSAGE"
90 PRINT "A""B"
100 PRINT """QUOTE"""
199 END
```

will cause the following display:

```
MESSAGE
A"B
"QUOTE"
```

6.7.3 Information Entry Statements (DATA, READ)

The DATA and READ statements are always used together. The DATA statement is used to set up a list of numeric or string values. These values are accessed by the READ statement which assigns those values to variables in a program.

6.7.3.1 DATA Statement Format – The DATA statement sets up a list of values to be used by the READ statement.

DATA Statement Form

Line Number	Statement	Parameters
xxx	DATA	Values

where values are numeric and/or string entries separated by commas. The DATA statement serves as a source of input variables for the program. The variables are accessed for processing by the READ statement.

There may be any number of DATA statements in a program. They are not executed and may be placed anywhere within the program. However, BASIC treats them all together as a single list.

Both string data and numeric data may be intermixed in a single DATA statement.

String data in a DATA list must always be enclosed by quotation marks.

For example, the three DATA statements

```
30 DATA "FIRST"
120 DATA 2, "SECOND"
240 DATA 3, "THIRD", 4
```

are equivalent to the following statement:

```
240 DATA "FIRST",2,"SECOND",3,"THIRD",4
```

6.7.3.2 READ Statement Format – The READ statement accesses variables defined by the DATA statement and assigns those values to variables in a program.

READ Statement Form

Line Number	Statement	Parameters
xxx	READ	variables

where variables are names corresponding to values contained in DATA statements. The following examples illustrate the use of the READ statement.

Variable names must be separated by commas.

A READ statement may contain numeric and string variable names intermixed, to correspond to numeric and alphanumeric values in the DATA list. However, since values are taken from the DATA list in sequential order, you must insure that the values in the list are in the correct sequence to correspond to the variable names of the same format (numeric or string).

A READ statement may have more or fewer variables than there are values in any one DATA statement. The READ statement causes BASIC to search all available DATA statements in the order of their line numbers until values are found for each variable in the READ. A second READ statement will begin reading values where the first one stopped.

The READ statement is always used in combination with the DATA statement.

Example 1:

The following lines

```
10 READ A,B,C
20 DATA 1,2,3
```

will set variable A equal to 1, B equal to 2, and C equal to 3.

Example 2:

These statements

```
50 READ C,D$,E,F$(K)
60 DATA 5, "AAA",12, "WORD"
```

will set:

C=5
D$="AAA"
E=12
F=$="WORD"

Example 3:

The following program will display the first and third variables in a DATA list:

```
50 READ A$,B$,C$
60 PRINT A$,C$
70 DATA "DIS","XXX","PLAY"
99 END
```

The screen will show:

```
DISPLAY
```

Example 4:

The following program uses DATA statements to supply both a variable number of scores and variable score values to an average calculation routine.

```
100 PRINT "NUMBER"
110 PRINT "OF SCORES","AVERAGE"
120 PRINT
125 READ N1
127 FOR I=1 TO N1
130 READ N
135 IF N=0 GOTO 999
140 LET S=0
150 FOR K=1 TO N
160 READ T
170 LET S=S+T
180 NEXT K
190 PRINT N,S/N
200 NEXT I
890 DATA 5
900 DATA 3,82,88,97
910 DATA 5,66,78,71,82,75
920 DATA 4,82,86,100,91
930 DATA 4,72,82,73,82
940 DATA 6,61,73,67,80,84,79
999 END

RUNNH

NUMBER
OF SCORES AVERAGE

 3       89
 5       74.4
 4       89.75
 4       77.25
 6       74

 READY
```

6.7.4 LET Statement

The LET statement assigns the value of an algebraic expression to a variable. Use of the word LET is optional.

LET Statement Form

Line Number	Statement	Parameters
xxx	LET	v = expression

where

v is a variable; and

expression is a number, another variable, a string (enclosed in quotes), or an arithmetic expression.

The following examples illustrate the use of the LET statement.

Example 1:

The numeric variable A is set equal to 5 by the statement

```
10 LET A = 5
```

Example 2:

The string variable A$ is set equal to "XYZ" by the statement

```
10 A$ = "XYZ"
```

Example 3:

An element (3,2) in array A is set equal to an element (1,4) in array B by the statement

```
10 LET A(3,2) = B(1,4)
```

NOTE

The LET statement does not necessarily imply an equality. LET means "evaluate the expression to the right of the equal sign and assign this value to the variable on the left". Thus, the statement

```
L=L+1
```

means "set L equal to a value one greater than it was before".

6.7.5 Loops (FOR and NEXT Statement)

6.7.5.1 FOR Statement Format – The FOR statement is used in combination with the NEXT statement to specify program loops.

FOR Statement Form

Line Number	Statement	Parameters
xxx	FOR	V = x to y [STEP z]

LOOP INDEX	INDEX INITIAL VALUE	INDEX TERMINAL VALUE	INDEX STEP VALUE

where

V is a variable that serves as the INDEX for the loop. The index value is incremented (or decremented) each time the loop is executed.

x is an expression (numerical value, variable name, or mathematical expression) indicating the initial value of the index, that is, the value it will have before the loop is executed the first time;

y is the terminal value of the index. When v is "beyond" y, the loop will not be repeated; and

z is the step value, that is, it is the value that is added to the index each time the loop is executed. Variable z may be assigned a minus value in cases where it is desired to decrement the index. If "STEP Z" is omitted, a value +1 is assumed.

The x, y, and z values are expressions: these expressions are evaluated upon first encountering the loop. This includes setting the index equal to the initial value. Therefore, the program can later jump back to the same FOR statement any number of times to re-start the loop. If the x, y, and z values are unchanged, the loop will be repeated the same number of times each time the program executes the FOR statement.

A variable used as an index in a FOR statement must not be subscripted.

The block of instructions to be executed repeatedly will immediately follow the FOR statement. After the last of these instructions there must be a NEXT statement whose parameter is the same as the index of the loop.

If the initial value is "beyond" the terminal value, the loop will never execute because an initial check is made of the starting and terminal values before the loop is executed. "Beyond", as used here, means that the initial value is not equal to the terminal value and adding the step value will only increase the difference. If the step value is negative "beyond" means "less than". If the step value is positive, " beyond" means "greater than".

The value of the loop index can be changed within the loop, thus influencing the number of times the loop is executed.

A program can have one or more loops within a loop. This is called "nesting loops", and is often used with subscripted variables.

It is possible to exit from a loop without the index reaching the terminal value by using an IF statement. Control may transfer into a loop only if that loop was left earlier without being completed.

The following examples illustrate the use of the FOR statement.

Example 1:

The following program

```
10 DIM C(2,3)
20 FOR A=1 TO 3
30 FOR B=1 TO 2
40 READ C(B,A)
50 PRINT C(B,A), B;A
60 NEXT B
70 NEXT A
80 DATA 1,2,3,4,5,6
90 END
```

will display:

```
1   1 1
2   2 1
3   1 2
4   2 2
5   1 3
6   2 3
```

Lines 30 through 60 of the above program represent a loop nested within another loop (line 20 through 70). The FOR statement on line 30 will be executed three times. Each time, its loop statements (lines 40 through 60) will be repeated twice.

Example 2:

The following lines

```
10 LET B=2\C=3
20 FOR A=C-B TO B^C STEP C
30 PRINT A
40 NEXT A
50 PRINT A
99 END
```

will display:

```
1
4
7
7
```

When the FOR statement was encountered, it interpreted the initial value as 1 (3 minus 2), the terminal value as 8 (2 to the third power) and the step value as 3.

Example 3:

The following loop

```
10 FOR I=10 TO 1 STEP -1
20 NEXT I
30 PRINT I
99 END
```

will display:

```
1
```

Example 4:

The following loop

```
10 FOR D=1 TO 5
20 LET D=D+4
30 NEXT D
99 END
```

will only be executed once.

Example 5:

The loops in the following program will never be executed:

```
10 FOR D=5 TO 1
20 PRINT D
30 NEXT D
10 FOR D=1 TO 5 STEP -1
20 PRINT D
30 NEXT D
99 END
```

6.7.5.2 NEXT Statement Format – The NEXT statement defines the end of a program loop.

NEXT Statement Form

Line Number	Statement	Parameters
xxx	NEXT	v (variable)

where v represents the index value used in the corresponding FOR statement.

A NEXT statement is never used without a FOR statement. The variable value must be identical to the index value (v) used in the corresponding FOR statement. The variable parameter may not be subscripted.

6.7.6 Control Statements (GOTO, IF-THEN, GOSUB, RETURN)

Normally the statements in a BASIC program are executed in the sequence they appear in a program. Control statements allow this execution sequence to be redirected.

6.7.6.1 Unconditional Branch (GOTO Statement) – The GOTO statement is used to transfer control to another line statement in a program. BASIC then continues execution at the line number referred to in the GOTO statement.

GOTO Statement Form

Line Number	Statement	Parameter
xxx	GOTO	n

where n is the line number of the statement to which control should be transferred.

For example, the following program employs two separate GOTO statements to redirect program control.

```
10 GOTO 40
20 PRINT "SECOND"
30 STOP
40 PRINT "FIRST"
50 GOTO 20
99 END
```

When executed, the program displays:

```
FIRST
SECOND
```

NOTE

When the program reaches the GOTO statement, the statements immediately following will not be executed; instead, execution is transferred to the statement beginning with the line number indicated.

6.7.6.2 Conditional Branch (IF-THEN Statement) – The IF-THEN statement tests a condition and redirects program control if that condition is true. That is, if the condition specified is true, the IF-THEN statement effectively executes a GOTO statement for the line number specified. When the condition is false, the next statement is executed.

Line Number	Statement	Parameters
xxx	IF	v1 relation v2 THEN n

where

v1, v2	are variables, numbers, strings, or expressions to be compared;
relation	is the relational operator to be used in comparing v1 and v2 (See Section 6.6.5.4);
THEN	may be replaced with GOTO if desired (either THEN or GOTO is acceptable, but one of the two must be present); and
n	is the number of the statement to which the program will jump if the relationship described is true.

The following examples illustrate the IF-THEN statement.

Example 1:

After executing the first two instructions

```
20 LET A=5
30 IF A =>2 GOTO 100
40 PRINT "NO"
100 PRINT "HERE"
```

control is transferred to line 100.

Example 2:

After executing the following instructions

```
20 LET A=5
30 IF A=2 GOTO 100
40 PRINT "NO"
99 END
```

the console will display:

```
NO
```

After executing the following instructions

```
10 LET A$="*"
20 IF A$= "Z" GOTO 99
30 PRINT "YES"
99 END
```

the program will display:

```
YES
```

because the ASCII code for "*" is not equal to the ASCII code ASCII for a "Z".

Strings may be used in IF-THEN statements, but comparisons are based on the positions of the characters in the string sequence. The question mark (?) is the highest alphanumeric character and the at sign (@) is the lowest.

The two strings described in the IF-THEN statement are compared one character at a time, from left to right, until the ends of the strings are reached or until an inequality is found.

If the strings are of unequal length, BASIC lengthens the shorter string by adding spaces to the right until both are of equal length. If "AB" is compared to a four-character string, it will be treated as "AB(space) (space".

When using numerical expressions in the IF-THEN statement, the test for equality may not always work due to the nature of the arithmetic used by the computer. One way to get around this is to compare the absolute value of the difference between the operands to a very small number. For example, instead of

```
20 IF A=B THEN 50
```

use

```
20 IF ABS(B) < 0001 THEN 50
```

6.7.6.3 Branch to Subroutine (GOSUB Statements) – A subroutine is a group of statements that perform a processing operation at more than one point in a program.

The GOSUB statement is used to branch to the subroutine and the RETURN statement redirects control back to the main body of the program.

GOSUB Statement Form

Line Number	Statement	Parameters
xxx	GOSUB	n

where n is the line number of the first line of the subroutine.

When the program encounters a GOSUB statement, the following action occurs:

1. BASIC internally records the number of the statement following the GOSUB statement.
2. Control is transferred to statement number n.

GOSUB is always used with the RETURN statement.

The RETURN statement is the last statement in a subroutine. When the program encounters the RETURN statement, control transfers back to the statement following the GOSUB.

A subroutine can call another subroutine. This is called "nesting" (see Example 2). Programs may be written to transfer control from one statement to another in the same subroutine, or to a statement in a different subroutine. When a RETURN is encountered, control returns to the statement following the last GOSUB that was executed. Subroutines may not be nested more than ten levels deep (Example 2 shows two levels of nesting).

The following examples illustrate the GOSUB statement.

Example 1:

The following program

```
100 LET A8="HI"
110 LET B$="THERE"
120 LET C$=" "
130 LET V$=A$
140 GOSUB 1000
150 LET V$=A$&C$&B$
160 GOSUB 1000
170 STOP
1000 REM PRINT V$
1010 PRINT V$
1020 RETURN
9999 END
```

will display:

```
HI
HI THERE
```

Example 2:

The following program, showing subroutine nesting,

```
100 FOR I=1 TO 3
110 LET V=I
120 GOSUB 1000
130 LET X=2*I
140 GOSUB 500
150 NEXT I
200 STOP

500 REM CALCULATE
510 LET V=X+2
520 GOSUB 1000
530 RETURN

1000 REM PRINT VALUE
1010 PRINT "THE VALUE IS";
1020 PRINT V
1030 RETURN
9999 END
```

will display:

```
THE VALUE IS 1
THE VALUE IS 4
THE VALUE IS 2
THE VALUE IS 6
THE VALUE IS 3
THE VALUE IS 8
```

6.7.6.4 Return from Subroutine (RETURN Statement) – The RETURN statement is used to redirect control from a subroutine.

RETURN Statement Form

Line Number	Statement	Parameters
xxx	RETURN	(none)

RETURN is always used with the GOSUB statement.

When the RETURN is encountered, it causes control to transfer to the statement following the last GOSUB executed.

6.7.7 Program Termination Statements (END, STOP)
BASIC is equipped with two statements, the END and STOP statements, that can be used to terminate program execution.

6.7.7.1 END Statement Format – The END statement terminates execution of the program. It informs the BASIC compiler that it has reached the last line of the program.

END Statement Form

Line Number	Statement	Parameters
xxx	END	None

NOTE
Only one END statement may appear in a program and it must be the last statement in the program.

6.7.7.2 STOP Statement Format – The STOP statement is used to terminate the execution of a program.

STOP Statement Form

Line Number	Statement	Parameters
xxx	STOP	None

Note that there may be several STOP statements in a program.

6.7.8 The INPUT Statement
The INPUT statement permits the operator to specify data during execution of a program.

INPUT Statement Form

Line Number	Statement	Parameters
xxx	INPUT	Variable List

where variables can be a single variable or a list of variables.

The INPUT Statement will cause the program to pause during execution, display a question mark, and wait for you to enter a value and press the RETURN key. If there are several variables involved, the program will expect you to type in a value corresponding to each variable. If you press the RETURN without having done this, the system will display another question mark and wait for the rest of the data. When the values have all been typed in, the program will continue with the variable names now equivalent to the values typed in. The first variable will equal the first entry, the second will equal the second entry, etc. (See Example 1).

The following values are recognized as acceptable when inputting numeric data:

+ or - sign
digits 0 through 9
the letter E
leading spaces (ignored)
(first decimal point)

All other characters are treated as delimiters for separating numeric data. That is, when the system encounters a character other than those specified, it will consider that it has come to the end of the entry relating to the variable it is currently processing and will apply any characters typed in after that to the following variable, if any (see Example 1).

When inputting numeric data, two delimiters read in succession imply that the data between delimiters is 0 (see Example 2).

In response to an INPUT statement, you can provide more values than are requested by the INPUT statement. The remaining or unused values are saved for subsequent use by the next INPUT statement. The question mark is not displayed until the program is out of data.

When inputting string data all characters are recognized as part of the string. Quotation marks are not typed in unless they are deliberately meant to be part of the string.

Each string requested by an INPUT statement must be terminated by a carriage return which acts as the data delimiter. This is necessary since all characters except for the carriage return are recognized as part of the data string.

String variables are assumed to be eight characters long unless otherwise described in a DIM statement.

The following examples illustrate the use of the INPUT statement.

Example 1:

If, in response to this statement:

```
100 INPUT A,B,C,D,E
```

you type

```
?-2,3.7A4E3 9<+1
```

the variables will have the following values:

A:-2
B:3.7
C:4000(4E3=4X10^3=4000)
D:9
E: 1(numbers are assumed to be positive unless they are specified to be negative).

The delimiters in the above line are the comma, the letter "A", the space after the number 3, the left-angle bracket (<), and the RETURN.

Example 2:

If, in response to the same statement, you type

```
?-2, 3.7, 4E3,,1
```

The results will be identical except that variable "D" will have the value 0.

Example 3:

If, in response to the statement:

```
50 INPUT R$
```

you type

```
?  "A,C>=+7
```

the string variable R$ will have the value:

"A,C>=+7

Example 4:

If, in response to:

```
40 LET A=5
50 INPUT B(A)
```

you type

```
? 7
```

B (5) will now have the value of 7.

6.7.9 The REMark Statement

The REM statement is a nonexecutable statement used to insert comments into the source program.

REMARK Statement Form

Line Number	Statement	Parameters
xxx	REM	comments

6.7.10 Ancillary Statements (DIMension, RESTORE, DEFine, RANDOMIZE and CHAIN)
BASIC has five additional statements that are used as helping statements and fall in no particular category. They are described in the following paragraphs.

6.7.10.1 DIMension Statement Format – The DIMension statement is used to describe subscripted variables and to define the length of strings.

DIMension Statement Form

Line Number	Statement	Parameters
xxx	DIM	v(n [,m])

where v is the name of the subscripted variable.

If the variable name (v) is numeric, and

1. m is omitted, then n+1 is the number of elements in an array or vector (see Example 1).
2. m is specified, then n+1 is the number of rows in a two-dimensional array (see Example 2).
3. if v is numeric, then m+1 is the number of columns in a two-dimensional array (see Example 2);

If the variable name (v) is alphanumeric, and

1. m is omitted, then n is the length of the string. This describes a single string, not a list, and cannot exceed 80 (see Example 3).
2. m is specified, then n+1 is the number of strings in the list. This is a one dimensional array (see Example 4).
3. if v is alphanumeric, then m is the length of each string in a one-dimensional list. It cannot exceed 80 (see Example 4).

The parameters n and m must be integer constants. They are limited in size only by the amount of available memory.

If a numeric variable is used in the program with a subscript but is not defined in a DIM statement, BASIC assigns it an array size of ten.

BASIC assumes a maximum string length of 8 characters unless the variable appears in a DIM statement.

Two-dimensional string variables are not permitted.

When the variable is used in other statements, it is not permitted to have subscripts whose values are higher than those in the DIM statement.

The first element of every array is automatically assumed to have a subscript of zero. Therefore, the number of boxes in a one dimensional array is n+1. The number of boxes in a two-dimensional array is (n+1)*(m+1). The first element in an array is v(0) or v(0,0) (see Example 2). However, the "zero" elements can be disregarded in programming unless the user wishes to conserve memory.

More than one array can be defined in a single DIM statement (see Example 7).

In general, wherever you can use a single variable name in a statement, you can use a subscripted one. Exceptions are noted in descriptions of the individual statements (such as the index of the FOR statement).

The following examples illustrate the use of DIMension statement.

Example 1:

The following statement

```
10 DIM A(5)
```

describes six numeric elements as follows:

A(0)	A(1)	A(2)	A(3)	A(4)	A(5)

To store a 5 in A(3) and then display it, type

```
20 LET A(3)=5
30 PRINT A(3)
```

Example 2:

The following statement

```
10 DIM A(3,5)
```

describes 24 numeric elements (4X6=24) as follows:

SIX COLUMNS

	A(0,0)	A(0,1)	A(0,2)	A(0,3)	A(0,4)	A(0,5)
FOUR	A(1,0)	A(1,1)	A(1,2)	A(1,3)	A(1,4)	A(1,5)
ROWS	A(2,0)	A(2,1)	A(2,2)	A(2,3)	A(2,4)	A(2,5)
	A(3,0)	A(3,1)	A(3,2)	A(3,3)	A(3,4)	A(3,5)

Example 3:

The following statement

```
10 DIM C$(12)
```

describes one string, 12 characters long:

C$

Example 4:

The following statement

```
10 DIM D$(3,20)
```

describes 4 strings, each 20 characters long:

D$(0) D$(1) D$(2) D$(3)

Example 5:

The following program will fill the array in Example 4 from a DATA list:

```
10 DIM D$(3,20)
20 FOR Y=0 TO 3
30 READ D$(Y)
40 NEXT Y
50 FOR Z=0 TO 3
60 PRINT D$(Z)
70 NEXT Z
80 DATA "ZERO","ONE","TWO","THREE"
99 END
```

and display each element:

```
ZERO
ONE
TWO
THREE
```

Example 6:

After these statements:

```
10 DIM B(3,5)
20 FOR G=1 TO 3
30 LET B(G,0)=G
40 NEXT G
50 FOR H=2 TO 5
60 LET B(0,H)=H
70 NEXT H
99 END
```

The area diagrammed in Example 2 would appear as follows:

			B(0,2)	B(0,3)	B(0,4)	B(0,5)
			2	3	4	5
B(1,0)	1					
B(2,0)	2					
B(3,0)	3					

Example 7:

The following statement dimensions both the one-dimensional array A and the two-dimensional array B:

```
10 DIM A(20), B(4,7)
```

6.7.10.2 RESTORE Statement – The RESTORE statement allows the program to go back to the beginning of a DATA list (paragraph 6.7.3.1) after using the list in a READ statement (paragraph 6.7.3.2).

RESTORE Statement Form

Line Number	Statement	Parameters
xxx	RESTORE	None

If it is desired to use the same data more than once in a program, RESTORE makes it possible to recycle through the DATA list beginning with the first value in the first DATA statement.

The RESTORE statement may be used in programs where DATA statements convey numeric or string data to READ statements.

The same variable names may be used the second time through the data since the values are being read as though for the first time.

The following example illustrates the use of the RESTORE statement.

The following lines

```
10 READ A,B,C,D
20 PRINT A;B;C;D
30 RESTORE
40 READ E,F,G,H
50 PRINT E;F;G;H
60 DATA 1,2,3,4
99 END
```

will cause this display:

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

6.7.10.3 DEFine Statement – This statement allows you to add functions to a program.

DEFine Statement Form

Line Number	Statement	Parameters
xxx	DEF	FNa(x)=expression

where

a is the capital letter used for identifying the function;

x is a dummy variable and must be the same on both sides of the equal (=) sign; and

expression defines the function by indicating the calculation process (function) involved.

There must be a DEF statement for each function used in the program.

The DEF statement must appear before the first use of the function it defines.

If there is more than one variable involved in the function, BASIC will identify them by their position.

Up to 14 different arguments may be used.

Up to 26 FN functions may be defined in a single program (FNA, FNB . . .FNZ).

The following examples illustrate the use of the DEFine statement.

Example 1:

The statement

```
10 DEF FND(S)=S^2
```

will cause the later statement

```
50 LET R=FND(4)
```

to be evaluated as R=16. BASIC locates the DEF statement for the function FND, substitutes the 4 for the variable (S) in the expression (S^2) and calculates the value of FND(4) to be 4^2=16.

NOTE

A variable that is used as a dummy argument in a DEF FNa statement can also be used elsewhere in the program.

Example 2:

This program:

```
10 DEF FNH(N,P)=2*P+N
20 LET X=4\LET Y=5
30 PRINT FNH(X,Y)
40 END
```

will display:

```
14
```

BASIC takes the first value in the function (4) as "N", because "N" appears first in the DEF statement. It takes the second value (5) as "P", because "P" is in the second position.

DEF FNH (N,P) = 2*P+N

first position second position

PRINT FNH(X,Y)

6.7.10.4 RANDOMIZE Statement – The RANDOMIZE statement is used with the RND function to generate a different set of numbers each time the program is run.

RANDOMIZE Statement Form

Line Number	Statement	Parameters
xxx	RANDOMIZE	None

6.7.10.5 CHAIN Statement – The CHAIN statement allows one program to execute another program. It can be used to divide large programs into a number of smaller programs that are to be written and stored separately.

CHAIN Statement Form

Line Number	Statement	Parameters
xxx	CHAIN	"dev:file.ex"

where "dev:file.ex" is the device and the file name of the program to be executed.

When BASIC encounters a CHAIN statement in a program, it stops execution of that program, retrieves the program named in the CHAIN statement from the specified device and file, compiles the CHAINed program (if necessary) and begins execution of that program.

If the program was started in the editor, when execution of all programs in the chain is complete, the workspace will contain the original program.

All output files that are opened in the original program must be closed before the CHAIN statement is encountered.

A BASIC language program may only CHAIN to another BASIC language program, and the program it CHAINs to may not have an extension of ".SV" (see the following example).

A compiled program may also execute CHAIN statements, but it can only CHAIN to other compiled programs which have ".SV" extensions.

In the following example, the program "SECOND" will CHAIN to the program "FIRST".

```
NEW FIRST

READY
10 PRINT "TARGET"
20 END
SAVE SYS:FIRST                    Enter FIRST and store it.

READY
NEW SECOND

READY
10 PRINT "ORIGINAL"
20 CHAIN "SYS:FIRST.BA"
30 END
SAVE SYS:SECOND                   Enter SECOND and store it.

READY
RUNNH                             Execute the programs.
ORIGINAL                          Displayed by SECOND.
TARGET                            Displayed by FIRST.
```

6.7.11 File Handling Statements

The file capability provided by OS/78 BASIC allows writing to or reading from the peripheral devices of the system.

6.7.11.1 FILE# Statement – The FILE# statement defines and opens a file.

FILE# Statement Form

Line Number	Statement	Parameters
xxx	FILE	t # n: "dev:file.ex"

where

t — must be one of the following:

(blank) – for an input string file
V – for an output string file
N – for an input numeric file
VN – for an output numeric file;

n — is the number you are assigning to the file (It must be 1, 2, 3, or 4 and can be a numeric variable); and

"dev: file.ex" — is the standard device, file name and extension; It must be either a string variable or the string itself in quotation marks.

A file must be opened before it can be used. The only exception is the terminal, which is always available for use.

The n in this statement is the number you must use in all FILE# statements that refer to the file. The terminal is always FILE#0.

The following examples illustrate the use of the FILE# statement.

Example 1:

The following statement describes file number 1 to be the string file HPRDAT.AS on RXA1 and opens it for output.

```
10 FILEV#1:"RXA1:HPRDAT.AS"
```

Example 2:

The following statement describes file number 2 to be the numeric file DATA.NU on RXA1 and opens it for output.

```
10 FILEVN#2:"RXA1:DATA.NU"
```

Example 3:

The following statement describes file number 3 to be the string file TEST.AB on RXA1 and opens it for input.

```
10 FILE#3:"RXA1:TEST.AB"
```

Example 4:

The following statement describes file number 4 to be the numeric file FILA.CD on RXA1 and opens it for input.

```
10 FILEN#4:"RXA1:FILA.CD"
```

6.7.11.2 **PRINT# Statement** – The PRINT# statement writes data into files.

PRINT# Statement Form

Line Number	Statement	Parameters
xxx	PRINT#	n:expressions

where

n	is the file number (It may be a numeric variable); and

expressions	depends on file type, numeric or string as discussed below.

As long as PRINT# is used for only numbers or numeric variables separated by commas or semicolons (or RETURN at the end of a PRINT# line), BASIC converts commas to spaces and does not write the carriage return and line feed to numeric files. The only thing written out will be a "list" of numbers separated by spaces. Each time INPUT# is used, it will read another number from the list (see Example 1).

When dealing with string files, symbols such as RETURNs, semicolons, and so forth, are used in the PRINT# statement in the same way they are used in the PRINT statement. The PRINT# statement works exactly the same way that the PRINT statement does, except that the line goes to the file designated instead of to the terminal.

The important difference here is that string files involve lines, while numeric files involve individual numbers. That is, each INPUT# will read a line (see Example 2).

If PRINT# is used for numerics to a string file, BASIC will convert them to strings. If an attempt is then made to INPUT# them into numeric variables, BASIC will convert them back to numerics. However, the RETURN and line feed at the end of a line will be converted to zeros. This can be dealt with by adding two extra variables to the INPUT# statement.

The following examples illustrate the use of the PRINT# statement.

Example 1:

The following lines

```
10 FILEVN#1:"SYS:TST.XX"
20 PRINT#1:1,2
30 PRINT#1:3,4,
40 PRINT#1:5,6
50 CLOSE#1
60 FILEN#1:"SYS:TST.XX"
70 FOR X=1 TO 6
80 INPUT#1:Z
90 PRINT Z
100 NEXT X
199 END
```

will display

```
1
2
3
4
5
6
```

Example 2:

The following lines

```
10 PRINT "A", "B"
20 PRINT "C"; "D";
30 PRINT "E"
40 END
```

will display

```
A           B
CDE
```

The same display will also be caused by

```
5 DIM J$(30)
10 FILEV#2:"RXA1:PROG.XX"
20 PRINT #2:"A", "B"
30 PRINT #2:"C"; "D";
40 PRINT #2:"E"
50 CLOSE#2
60 FILE#2:"RXA1:PROG.XX"
70 INPUT#2:J$
80 PRINT J$
90 INPUT#2:J$
100 PRINT J$
199 END
```

6.7.11.3 INPUT# Statement – The INPUT# statement reads data from a file.

INPUT# Statement Form

Line Number	Statement	Parameters
xxx	INPUT#	n:variables

where

n is the file number of the file being read (it may be a variable); and

variables is the list of variables into which data will be read. Each variable is separated by a comma.

Normally, data from numeric files is read into numeric variables and data from string files is read into string variables.

It is possible however to write numerics into a string file and then read them into either numeric or string variables, depending on how it is desired to use them. If numbers are read from a string file into string variables, they will be in string form and subject to the same rules as other strings.

Numbers read from a string file into numeric variables will be converted to numerics. Line feeds are ignored in this case.

The following examples illustrate the use of the INPUT# statement.

Example 1:

To read two strings from RXA1:FIL.DA, enter

```
10 FILE#1:"RXA1:FIL.DA"
20 INPUT#1:A$,B$
```

A$ will contain the first string, B$ the second string.

Example 2:

To read five numbers from RXA1:TST.XX, enter

```
10 FILEN#3:"RXA1:TST.XX"
20 INPUT#3:A,B,C,D,E
```

Example 3:

The following program writes numerics to a string file and reads them back as numerics:

```
10 FILEV#1:"SYS.FILA.ZZ"
20 FOR I=1 TO 5
30 PRINT#1:I
40 NEXT I
50 CLOSE#1
60 FILE#1:"SYS:FILA.ZZ"
70 FOR I=1 TO 5
80 INPUT#1:J,C,L
90 PRINT J
100 NEXT I
110 END
```

It will display:

```
1
2
3
4
5
```

6.7.11.4 RESTORE# Statement – The RESTORE statement resets the file data printer back to the beginning of the file, that is, RESTORE effectively performs a file close followed immediately by a file open so that the first data element in the file can be reread.

RESTORE# Statement Form

Line Number	Statement	Parameters
xxx	RESTORE#	n

where

n is the number of the file to be reset. It may be a number or a numeric variable.

For example, if RXA1:FILB.LM is a numeric file containing the numbers 1 through 9, the instructions

```
100 FILEN#3:"RXA1:FILB.LM"
110 FOR I=1 TO 3
120 INPUT #3:Z
130 PRINT Z
140 NEXT I
150 RESTORE#3
160 INPUT#3:Z
170 PRINT Z
199 END
```

will display:

```
1
2
3
1
```

NOTE
If n is zero, the DATA list in the program is reset.

6.7.11.5 CLOSE# Statement – The CLOSE# statement finishes the processing of a file and allows it number to be assigned to another file in a FILE# statement.

FILE CLOSE# Statement Form

Line Number	Statement	Parameters
xxx	CLOSE#	n

where

n is the number of the file to be closed. It may be a variable.

For example, in the following program

```
50 FILEV #1:"SYS:TEST.XX"
60 PRINT #1:"A","B","C","D"
70 CLOSE #1
80 FILE #1:"RXA1:FILD.DA"
90 INPUT #1:J$
99 END
```

The CLOSE# statement at line 70 finished the processing of file SYS:TEST.XX and allowed its number, 1, to be assigned to RXA1:FILD.DA in line 80.

NOTE
All output files must be CLOSED before any of the following is executed.

CHAIN	END
STOP	CTRL/C

Failure to do so results in loss of file.

6.7.11.6 IF END# Statement – The IF END# statement determines if the End of File (EOF) marker has been read from a file, and if so, branches to the line specified.

IF END# Statement Form

Line Number	**Statement**	**Parameters**
xxx	IF END#	n THEN m

where:

n is the file number of the file in question (it may be a variable); and

m is the line number to which control passes when the end of the file is detected.

This statement works only for string files.

The IF END# statement should come immediately after the PRINT# or INPUT# statement for that file. If, as a result of the IF END# statement, control passes to line m, it means that the last PRINT# or INPUT# was not successful. That is, nothing was actually read from or written to the file as a result of the last INPUT# or PRINT# statement.

For Example,

The following lines

```
 10 FILEV#1:"SYS:PROGA.BB"
 20 PRINT#1:"A"
 30 PRINT#1:"B"
 40 CLOSE#1
 50 FILE#1:"SYS:PROGA.BB"
 60 INPUT#1:A$
 70 IF END#1 THEN 100
 80 PRINT A$
 90 GOTO 60
100 PRINT "END OF FILE"
110 CLOSE#1
199 END
```

will display

```
A
B
END OF FILE
```

6.8 BASIC FUNCTIONS

BASIC functions are standard subroutines incorporated into the BASIC Run Time System (BRTS) to aid computations and text handling.

Function calls consist of a three letter (all capitals) name followed by an argument in parentheses. The argument may be a number, variable, expression, or another function. Generally, functions may be used anywhere a number or variable is legal in a mathematical expression.

Most functions compute a value based on the value of the argument or arguments involved. They are said to return this value. For example, SQR(A) "returns" the square root of A.

Functions may return either strings or numbers. Functions that return strings have names ending in a dollar sign (STR$, SEG$), while functions returning numbers have names that do not end in a dollar sign (SGN, VAL).

6.8.1 Arithmetic Functions (ABS, INT, EXP, RND, SGN, SQR)

6.8.1.1 BASIC ABS Function – The ABS function returns the absolute value of an expression.

Format

ABS(X)

where

X is a number, numeric variable, or numeric expression

6.8.1.2 BASIC EXP Function – The EXP function calculates the value of e raised to the X power, where e is equal to 2.71828. That is, EXP(X) is equivalent to 2.71828 X.

Format

EXP(X)

where

X is a number, numeric variable, expression, or another function.

6.8.1.3 BASIC INT Function – The INT function returns the value of the largest integer not greater than the argument.

Format

INT(X)

where

X is a number, numeric variable, expression, or another function.

NOTE

This function can be used to round numbers to the nearest integer by specifying INT(X+.5).

For example, the function INT(34.67) has the value 34; the functions INT(34.67+.5) and INT(34.37+.5) have these values of 35 and 34, respectively; and these functions INT(-23) and INT(-14.39) have these values of -23 and -15, respectively.

6.8.1.4 BASIC RND Function – The RND function produces random numbers between (but not including 0 and 1.

Format

RND(X)

where

X is a dummy variable in this function.

Every time this function is encountered in a statement, it will produce a different set of decimal numbers. However, the program is RUN again, the same set of numbers will be produced (see Example 1).

If this repetition is undesirable, it can be changed with the RANDOMIZE statement.

For example, the following program is run twice with identical results:

```
10 FOR A = 1 TO 5
20 PRINT RND(X)
30 NEXT A
40 END

RUNNH
0. 361572
0. 332764
0. 633057
0. 350342
0. 670166

READY
RUNNH
0. 361572
0. 332764
0. 633057
0. 350342
0. 670166

READY
```

6.8.1.5 **BASIC SGN Function** – The SGN function creates a value based on the sign of the argument.

Format

SGN(X)

where

X is a number, numeric variable, numeric expression, or another function.

NOTE

The value of the SGN function will be 1 if the argument is any positive number, 0 if the argument is zero, and - 1 if the argument is negative.

6.8.1.6 **BASIC SQR Function** – The SQR function computes the positive square root of an expression.

Format

SQR(X)

where

X is a number, numeric variable, numeric expression, or another function.

NOTE

If the argument is negative, the absolute value of the argument is used.

6.8.2 Trigonometric Functions (ATN, COS, LOG, SIN)

6.8.2.1 BASIC ATN Function – The ATN function calculates the angle (in radians) whose tangent is given as the argument of the function.

Format

```
ATN(X)
```

where

X   is a number, numeric variable, expression, or another function, representing the tangent of an angle.

6.8.2.2 BASIC COS Function – The COS function is used to calculate the cosine of an angle specified in radians.

Format

```
COS(X)
```

where

X   is a number, numeric variable, expression, or another function, representing the size of an angle in radians.

6.8.2.3 BASIC LOG Function – The LOG function calculates the natural logarithm of X (to the base e).

Format

```
LOG(X)
```

where

X   is a number, numeric variable, expression, or another function.

6.8.2.4 BASIC SIN Function – The SIN function is used to calculate the sine of an angle specified in radians.

Format

```
SIN(X)
```

where

X   is a number, numeric variable, expression or another function, representing the size of an angle in radians.

6.8.3 String Handling Functions (ASC, CHP$, DAT$, LEN, POS, SEG$, STR$, TAB, VAL)

6.8.3.1 BASIC ASC Function – The ASC function converts a one character string to its code number (see CHR$).

Format

ASC(X)

where

X   is a one character string.

To find what will be returned for any character, look for the character in the “CHARACTER” column of Table 6-1. The number to the left of it is the decimal equivalent.

Table 6-1 Decimal/Character Conversions

Decimal	Character	Decimal	Character
0	@	32	(space)
1	A	33	!
2	B	34	”
3	C	35	#
4	D	36	$
5	E	37	%
6	F	38	&
7	G	39	'
8	H	40	(
9	I	41	)
10	J	42	*
11	K	43	+
12	L	44	,
13	M	45	-
14	N	46	.
15	O	47	/
16	P	48	0
17	Q	49	1
18	R	50	2
19	S	51	3
20	T	52	4
21	U	53	5
22	V	54	6
23	W	55	7
24	X	56	8
25	Y	57	9
26	Z	58	:
27	[	59	;
28	\	60	<
29	]	61	=
30	^	62	>
31	_	63	?

For example,

The following program

```
10 LET A$="*"
20 PRINT ASC("P"), ASC(A$),ASC("9")
30 END
```

will display

```
16      42      57
```

6.8.3.2 BASIC CHR$ Function – The CHR$ function converts a code number (modulo 64) to its equivalent character in the 64 character set.

Format

CHR$(X)

where

X is a number, numeric expression, or numeric variable (0 < X < 63).

To find what will be returned for any number, look for the number in the "DECIMAL" column of the preceding conversion table. The equivalent character will be to the right of the number.

For example, the following line

```
10 PRINT CHR$(1),CHR$(40)
```

will display:

```
A    (
```

6.8.3.3 BASIC DAT$ Function – The DAT$ function returns the current system date.

Format

DAT$(X)

where

X is a dummy variable in this function.

The date is returned as an eight-character string of the form:

MM/DD/YY

If the date has not been specified with the Monitor DATE command, no characters will be returned.

For example, the following lines

```
10 LET D$ = DAT$(X)
20 PRINT D$
```

will display:

```
07/20/77
```

if that date was entered with the Monitor DATE command.

6.8.3.4 BASIC LEN Function – The LEN function returns the number of characters in a string.

Format

LEN(X$)

where

X$ is a string or string variable. It may be several concentrated strings and/or variables.

6.8.3.5 BASIC POS Function – The POS function returns the location of a specified group of characters in a string.

Format

POS(X$, Y$, Z)

where

X$ is the string to be searched (it may be a string variable or string constant);

Y$ is the series of characters you are searching for (it may be a string variable or a string constant); and

Z is the position in the string at which you want to begin the search.

This function searches X$ for the first occurrence of Y$. The search begins with the Zth character in X$.

Z may not be less than zero or greater than the length of string X$.

If Y$ contains no characters, the function returns a one.

If X$ contains no characters, it returns a zero.

If Y$ is not found, it returns a zero.

For example, the following lines

```
10 DIM B2$(12)
20 B2$ = "ABCDEFGHIDEF"
30 PRINT POS (B2$, "DEF", 7)
```

will display:

```
10
```

6.8.3.6 BASIC SEG$ Function – The SEG$ function returns the sequence of characters between two positions in a string.

Format

SEG$(X$,Y,Z)

where

X$	is the string containing the characters to be returned (it may be a string variable or a string constant);
Y	is the position of the first character to be returned; and
Z	is the position of the last character to be returned.

If Y is less than 1, it is set to 1.

If Y is greater than the length of X$, no characters are returned.

If Z is less than 1, no characters are returned.

If Z is greater than the length of X$, it is set equal to the length of X$.

If Z is smaller then Y, no characters are returned.

For example, the following lines

```
10 DIM B2$(12)
20 B2$ = "ABCDEFGHIDEF"
30 PRINT SEG$(B2$,3,5)
```

will display:

```
CDE
```

6.8.3.7 BASIC STR$ Function – The STR$ function converts numbers to strings.

Format

STR$(X)

where

X	is a numeric expression

NOTE

The string that is returned is in the form in which numbers are output in BASIC without leading or trailing blanks.

6.8.3.8 BASIC TAB Function – The TAB function allows you to position characters anywhere on the terminal line.

Format

TAB(X)

where

X is the position (from 1 to 80) in which the next character will be displayed.

This function may only be used in a PRINT or PRINT# statement.

Positions on the line are considered by BASIC to be numbered from 1 to 80 across the screen from left to right.

Each time the TAB function is used, positions are counted from the beginning of the line, not from the current position of the cursor.

If X is less than the current position of the cursor, the display starts at the current position.

If X is greater than 80, the display will begin at the first position of the next line.

In order to keep track of the cursor, BASIC maintains a "column count", which represents the position of the cursor at any given time. As the cursor moves across the screen, BASIC adds to the column count. When the cursor returns to the first position, the column count is reset to 0. The activity of the TAB function is based on this count.

There are circumstances in which the column count does not coincide with the position of the cursor. For example, the PNT(07) function will add 1 to the count without moving the cursor. Also, the PNT(13) function will return the cursor to the first position on the screen without setting the column count to zero. The user, therefore, must take this into account when using the TAB function after PNT functions. The column count will be corrected the next time there is a "normal" return to column one.

For examples, the following lines

```
60 LET B=5
70 PRINT "A";TAB(B);"C"
```

will cause this display:

```
A   C
```

6.8.3.9 **BASIC VAL Function** – The VAL function converts a string to numeric data.

Format

VAL(X$)

where

X$ is a string constant or string variable made up of those values that BASIC recognizes as acceptable when inputting numeric data:

+ or - sign
digits 0 through 9
the letter E
leading spaces (ignored)
. (first decimal point)

NOTE

A string, even though it is composed of digits, is not numeric data. It cannot be used in calculations or as the argument of a mathematical function (SQR, ABS, EXP, and so forth), without first being converted by the VAL function.

6.8.4 Display Console Control Function (PNT)

BASIC PNT Function – The PNT function is used to perform special actions on the terminal, such as sounding the buzzer, erasing the screen, and moving the cursor.

Format

PNT(X)

where

X is the value of the character to be output.

Special actions that can be performed by the PNT function are as follows:

PNT(07)	sounds buzzer
PNT(08)	moves cursor one space to left
PNT(09)	moves to next tab stop (Tab stops are set every 8 spaces.)
PNT(10)	moves cursor down one line and scrolls if required
PNT(13)	moves cursor to left margin of current line
PNT(27);"A"	moves cursor up one line
PNT(27);"C"	moves cursor right one position
PNT(27);"H"	moves cursor to upper lefthand corner of screen ("home" position)
PNT(27);"J"	erases from cursor position to end of screen
PNT(27);"K"	erases line from cursor to right margin
PNT(27);"["	stops display of any new lines when screen is full. Each time the operator presses the SCROLL key, another line will be displayed. If he presses SHIFT and SCROLL, twelve new lines will be displayed. This is called Hold Screen Mode.
PNT(27);CHR$(28)	turns off Hold Screen Mode.

NOTE

The PNT function may only be used in a PRINT or PRINT# Statement.

6.8.5 Trace Function

BASIC TRC Function – The TRC function causes BASIC to print the line number of each statement in the program as it is executed.

Format

v = TRC(X)

where

v   can be any letter (it is a dummy argument and has no purpose except to occupy that position on the line); and

x   is 1 to turn the function on and 0 to turn it off.

This function is used to follow the progress of a program and help in tracking down errors.

When BASIC encounters TRC(1) in a program, it displays the line number of each line in the program as it is executed. The line numbers are displayed between percent signs.

When TRC(0) is encountered, the function is turned off and normal program operation resumes.

Certain types of statements are not recorded by TRC: namely, DATA, DEF, DIM, END, GOTO, NEXT, RANDOMIZE, REM, and STOP.

For example, the following program

```
60 T=TRC(1)
70 GOSUB 90
80 GOTO 140
90 PRINT "IN OUTER SUB"
100 GOSUB 120
110 RETURN
120 PRINT "IN INNER SUB"
130 RETURN
140 T= TRC(0)
150 END
```

will display:

```
% 70 %
% 90 %
IN OUTER SUB
% 100 %
% 120 %
IN INNER SUB
% 130 %
% 110 %
% 140 %
```

6.9 SUMMARY OF BASIC EDITOR COMMANDS

Command	Function
BYE	Exits from the editor and returns control to the monitor
LIst	Displays the program statements in the workspace with a header
LISTNH	Displays the program statements in the work space, without a header
NAme	Renames the program in the workspace
NEW	Clears the workspace and tells the editor the name of the program the user is about to type
OLd	Clears the workspace, finds a program on the disk, and puts in into the workspace
RUn	Executes the program in the workspace, after displaying a header
RUNNH	Executes the program in the workspace, without displaying a header
SAve	Puts the program in the workspace on a disk
SCratch	Erases all statements from the workspace

6.10 SUMMARY OF BASIC STATEMENTS

Statement	Function
CHAIN	Executes another program Example: 40 CHAIN "SYS:PROG.BA"
CLOSE#	Closes a file Example: 100 CLOSE#1
DATA	Sets up a list of values to be used by the READ statement Example: 240 DATA "FIRST",2,3
DEF	Defines functions Example: 10 DEF FND(S)=S+5
DIM	Describes a string and/or any subscripted variables Example: 50 DIM B(3,5),D$(3,72)
END	Terminates program compilation and execution Example: 100 END
FILE#	Defines and opens a file Example: 20 FILEVN#2:"RXA1:DATA.NV"

Statement	Function
FOR	Describes program loops (used with NEXT) Example: 60 FOR X=1 TO 10 STEP 2
GOSUB	Transfers control to a subroutine (used with RETURN) Example: 50 GOSUB 100
GOTO	Transfers control to another statement Example: 100 GOTO 50
IF	Tests the relationship between two variables, numbers, or expressions Example: 20 IF A=0 THEN 50
IF END#	Tests for the end of a string file Example: 60 IF END#3 THEN 100
INPUT	Accepts data from the terminal Example: 80 INPUT A,B,C
INPUT#	Reads data from a file Example: 50 INPUT#1:A$
LET	Assigns a value to a variable Example: 90 LET A$="XYZ"
NEXT	Indicates the end of a program loop (used with FOR) Example: 140 NEXT I
PRINT	Displays data on the screen Example: 200 PRINT A,"X";6
PRINT#	Writes data to a file Example: 180 PRINT#1:J
RANDOMIZE	Causes the RND function to produce a different set of numbers each time the program in run Example: 10 RANDOMIZE
READ	Sets variables equal to the values in DATA statements Example: 50 READ A$,B
REM	Inserts comments into the program Example: 30 REM COMPUTE EARNINGS
RESTORE	Sets program READ statements back to the beginning of the DATA list Example: 85 RESTORE

Statement	Function
RESTORE#	Resets a file pointer back to the beginning of that file Example: 130 RESTORE#3
RETURN	Returns control from a subroutine (used with GOSUB) Example: 115 RETURN
STOP	Terminates program execution Example: 40 STOP

6.11 SUMMARY OF BASIC FUNCTIONS

Command	Function
ABS(X)	Returns the absolute value of an expression Example: 10 LET X=ABS(-66) will assign X a value of 66
ASC(X$)	Converts a one character string to its code number Example: 20 PRINT ASC("B") will display 2
ATN(X)	Calculates the angle (in radians) whose tangent is given as the argument Example: 30 LET X=ATN(.57735) will assign X a value of 0.523598
CHR$(X)	Converts a code number to its equivalent character Example: 40 PRINT CHR$(1) will display A
COS(X)	Returns the cosine of an angle specified in radians Example: 50 LET Y=COS(45*3.14159)/180 will assign Y a value of 0.707108
DAT$(X)	Returns the current system date Example: 60 PRINT DAT$(X) will display the system date, such as 07/20/77
EXP(X)	Calculates the value of e raised to a power, where e is equal to 2.71828 Example: 30 IF Y>EXP(1.5) GOTO 70 will go to line 70 if Y is greater than 4.48169
INT(X)	Returns the value of the nearest integer not greater than the argument Example: 60 LET X=INT(34.67) will assign X the value 34

Command	**Function**
LEN(X$)	Returns the number of characters in a string Example: 10 PRINT LEN ("DOG") will display 3
LOG(X)	Calculates the natural logarithm of the argument Example: 10 PRINT LOG(959) will display 6.86589
PNT(X)	Outputs non-printing characters for terminal control Example: 50 PRINT PNT(13) will move the cursor to the left margin of the current line
POS(X$,Y$,Z)	Returns the location of a specified group of characters (Y$) in a string (X$) starting at a character position (Z) Example: 60 LET V= POS("ABCDBC", "BC",4) will assign V a value of 5
RND(X)	Returns a random number between (but not including) 0 and 1 Example: 70 PRINT RND(X) will display a decimal number, such as 0.361572
SEG$(X$,Y,Z)	Returns the sequence of characters in a string (X$) between two positions in the string (X,Y) Example: 30 LET R$=SEG$("ABCDEF",2,4) will assign R$ a value of BCD
SGN(X)	Returns 1 if the argument is positive, 0 if it is zero, and -1 if it is negative Example: 200 PRINT 5*SGN(-6) will display -5
SIN(X)	Returns the sine of an angle specified in radians Example: 30 LET B=SIN(30*3.14159/180) will assign B a value of 0.5
SQR(X)	Returns the positive square root of an expression Example: 40 PRINT SQR(16) will display 4
STR$(X)	Converts a number into a string Example: 120 PRINT STR$(1.76111124) will display the string 1.76111

Command	Function
TAB(X)	Positions characters on a line Example: 70 PRINT "A";TAB(5);"B" will display A B
TRC(1)	Causes BASIC to display the line number of each statement in the program as it is executed Example: 10 V=TRC(1) will display the line number of each statement executed until a TRC (0) is encountered
VAL(X$)	Converts a string to a number Example: 90 PRINT VAL("2.46111")*2 will display 4.92222

6.12 BASIC ERROR MESSAGES

6.12.1 Compiler Error Messages

The following error messages are generated by the BASIC compiler:

CH	ERROR IN CHAIN STATEMENT	NM	MISSING LINE NUMBER
DE	ERROR IN DEF STATEMENT	OF	OUTPUT FILE ERROR
DI	ERROR IN DIM STATEMENT	PD	PUSHDOWN STACK OVERFLOW
FN	ERROR IN FILE NUMBER OR NAME	QS	STRING LITERAL TOO LONG
FP	INCORRECT FOR STATEMENT	SS	BAD SUBSCRIPT OR FUNCTION ARG
FR	ERROR IN FUNCTION ARGS	ST	SYMBOL TABLE OVERFLOW
IF	ERROR IN IF STATEMENT	SY	SYSTEM INCOMPLETE
IC	I/C ERROR	TB	PROGRAM TOO BIG
LS	MISSING EQUALS SIGN IN LET	TD	TOO MUCH DATA IN PROGRAM
LT	STATEMENT TOO LONG	TS	TOO MANY CHARS IN STRING
MD	MULTIPLY DEFINED LINE NUMBER	UD	ERROR IN UDEF STATEMENT
ME	MISSING END STATEMENT	UF	FOR STATEMENT WITHOUT NEXT
MO	OPERAND EXPECTED, NOT FOUND	US	UNDEFINED STATEMENT NUMBER
MP	PARENTHESIS ERROR	UU	USE STATEMENT ERROR
MT	OPERAND OF MIXED TYPE	XC	CHARS AFTER END OF LINE
NF	NEXT STATEMENT WITHOUT FOR		

6.12.2 Run-Time System Error Messages

The following error messages are generated by the BASIC run-time system:

BO	NO MORE BUFFERS AVAILABLE	GS	TOO MANY NESTED GOSUBS
CI	IN CHAIN, DEVICE NOT FOUND	IA	ILLEGAL ARG IN UDEF
CL	IN CHAIN, FILE NOT FOUND	IF	ILLEGAL DEV:FILENAME
CX	CHAIN ERROR	IN	INQUIRE FAILURE
DA	READING PAST END OF DATA	IO	TTY INPUT BUFFER OVERFLOW
DE	DEVICE DRIVER ERROR	LM	TAKING LOG OF NEGATIVE NUMBER
DC	NO MORE ROOM FOR DRIVERS	OE	DRIVER ERROR WHILE OVERLAYING
DV	ATTEMPT TO DIVIDE BY ZERO	OV	NUMERIC OR INPUT OVERFLOW
EF	LOGICAL END OF FILE	PA	ILLEGAL ARG IN POS
EM	NEGATIVE NUMBER TO REAL POWER	RE	READING PAST END OF FILE

EN	ENTER ERROR
FB	USING FILE ALREADY IN USE
FC	CLOSE ERROR
FF	FETCH ERROR
FI	CLOSING OR USING UNOPENED FILE
FM	FIXING NEGATIVE NUMBER
FN	ILLEGAL FILE NUMBER
FO	FIXING NUMBER > 4095
GR	RETURN WITHOUT GOSUB
SC	CONCATENATED STRING TOO LONG
SL	STRING TOO LONG OR UNDEFINED
SR	READING STRING FROM NUMERIC FILE
ST	STRING TRUNCATION ON INPUT
SO	SUBSCRIPT OUT OF RANGE
SW	WRITING STRING INTO NUMERIC FILE
VR	READING VARIABLE LENGTH FILE
WE	WRITING PAST END OF FILE

CHAPTER 7
OS/78 FORTRAN IV

7.1 SYSTEM OVERVIEW

OS/78 FORTRAN IV provides the full capability of the FORTRAN IV language. In describing OS/78 FORTRAN, this chapter assumes that you are familiar with the material previously presented on the OS/78 operating system.

FORTRAN IV may be invoked through the commands COMPILE, LOAD, and EXECUTE. (The functions of these commands are described in detail in Chapter 3.) In addition, your FORTRAN program usually reads and writes data files under the supervision of the run-time system; FORTRAN I/O files are treated separately in the section on the run-time system (Section 7.2.3).

7.1.1 Source Programs

An alphanumeric file containing FORTRAN IV statements is called a source program. Source programs are usually created with the OS/78 Editor (Figure 7-1).

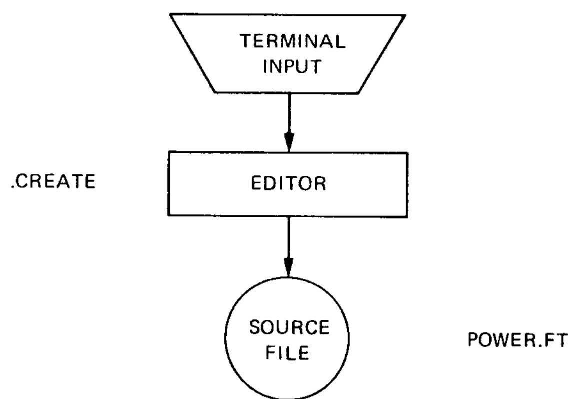


Figure 7-1 Creating a FORTRAN IV Program

7.1.2 Using the EXECUTE Command

A FORTRAN source program may be executed by successively (1) calling the compiler to convert the source program into a relocatable binary file, (2) calling the loader to link and relocate the binary file, and finally (3) calling the run-time system to load the program and supervise its execution. OS/78 FORTRAN IV provides a program chaining feature that can usually simplify or eliminate this sequence of explicit program calls. When chaining is invoked, the first system program to be executed automatically calls the second program in the compilation/loader/run-time system sequence and so forth. In this manner, simple FORTRAN programs may be compiled, loaded, and executed, all as the result of typing the single command EXECUTE (Figure 7-2).

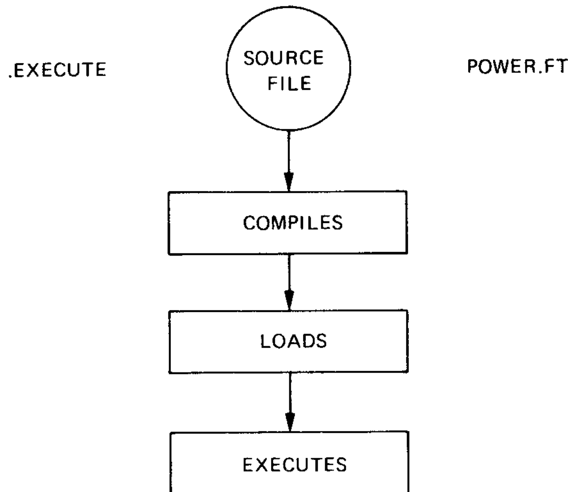


Figure 7-2 Executing a FORTRAN IV Program

FORTRAN programs containing subroutines cannot be adapted to the chaining facility because they require additional input specifications at some point. In general, however, it is usually most convenient to chain from the loader to the run-time system (combining relocation, loading, and execution).

Errors encountered by the various system programs do not result in the termination of program chaining unless the error is such that it is impossible for execution to continue. This permits the system to locate and identify as many errors as possible before returning control to the Monitor. When chaining is requested, intermediate output files produced by one system program are automatically deleted after they have been read as input by the next program in the chain sequence. This optimizes storage requirements.

7.1.3 Compiling

Once a source program has been prepared, it can be input to the compiler which translates each FORTRAN statement into one or more relocatable instructions (see Figure 7-3).

Compilation is accomplished in three passes. System program F4.SV begins compilation by building a symbol table and generating intermediate code. F4 chains to PASS2.SV automatically, and PASS2 calls PASS2O.SV to complete the translation into a special assembly language. If a source listing was requested, PASS2O chains to PASS3.SV automatically, and PASS3 generates the listing. PASS2, PASS2O and PASS3 may not be accessed directly.

The compiler output produced is then automatically assembled by the RALF Post-processor which completes the translation of FORTRAN IV source statements into relocatable computer instructions.

7.1.4 Loading

The relocatable binary file produced by compilation is a machine language version of a single program or subroutine. This file must be linked with its main program (if it is a subroutine) and with any other subroutines, including system library subroutines (FORLIB.RL) that it needs in order to execute. The system program LOAD.SV (the FORTRAN IV loader) accepts a list of relocatable binary file specifications from the console terminal and builds a loader image file containing a relocated main program linked to library components required for execution. (See Figure 7-4.)

The loader image file is executable except for run-time I/O specifications. It may be stored on any mass storage (directory) device and executed whenever desired. The loader also produces an optional symbol map that indicates the memory storage requirements of the linked and relocated program.

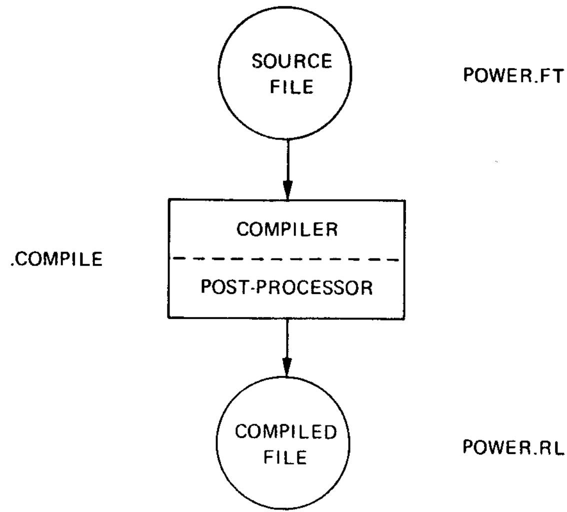


Figure 7-3 Compiling a FORTRAN IV Program

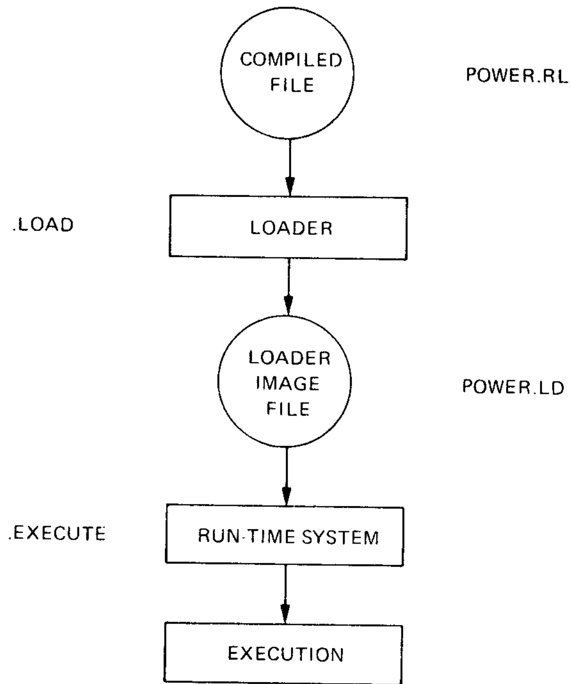


Figure 7-4 Loading and Executing a FORTRAN IV Program

7.1.5 Run-Time System

The loader image file produced by the FORTRAN IV loader is read and executed by system program FRTS.SV (the FORTRAN IV run-time system). This program also configures an I/O supervisor to handle any FORTRAN input or output in accordance with run-time I/O specifications. This arrangement makes the full I/O device independence of the OS/78 operating system available to every FORTRAN IV program, and permits FORTRAN programs to be written without concern for, or even knowledge of, the hardware configuration on which they will be executed.

The run-time system assigns I/O device handlers to the I/O unit numbers referenced by the FORTRAN program, allocates I/O buffer space, and also diagnoses certain types of errors that occur when the loader image file is read into memory. If no load time errors are encountered, the run-time system starts the FORTRAN program and monitors execution to check for run-time errors such as data I/O, numeric overflow, and hardware malfunctions. Run-time errors are identified at the console terminal, and, when a run-time error occurs, the system also provides an error traceback to identify the full sequence of FORTRAN statements that terminated in the error condition.

The compiler, loader, and run-time system each accept standard OS/78 option specifications. The option specifications are alphanumeric characters that may be thought of as switches whose presence or absence enables or disables certain program features and conventions. For example, specifying the /N option to the compiler suppresses compilation of internal statement numbers (described in Section 7.2.1), thereby reducing program memory requirements (at the cost of disallowing full error traceback during execution).

7.1.6 System Library

The FORTRAN IV system also includes FORLIB.RL, a library of FORTRAN functions and subroutines. Almost every FORTRAN program executes calls to library functions and subroutines that perform such tasks as mathematical function evaluation, data I/O and numeric conversion. When the loader recognizes that a program or subroutine has called a library component, it copies a relocated version of the referenced library routine into the loader image file and links it to the calling routine.

7.2 OS/78 FORTRAN IV OPERATION

It is important to identify the three processes that must be performed in the proper sequence to execute a FORTRAN source program. These processes are compilation, relocation, and execution. A detailed description of each process is given in the following sections.

It is also important to identify the types of input that must be supplied to each process and the types of output, along with the standard FORTRAN IV extensions, that are produced.

Table 7-1 lists the standard file extensions used to identify various types of source and system-generated files.

Table 7-1 Standard FORTRAN IV File Extensions

Extension	File Type
.FT	FORTRAN language source file.
.RA	RALF assembly language file.
.RL	Relocatable binary (assembler output).
.LD	Loader image.
.LS	Listing.
.TM	System temporary file. Created by certain multipass programs and normally deleted automatically after use.
.MP	Load map listing file.

Table 7-2 briefly illustrates the OS/78 operational process used for FORTRAN IV, showing the input files, the process and the resulting output.

Table 7-2 OS/78 FORTRAN IV Operational Process

Input	Process	Output
FORTRAN Source Programs (FT)	COMPILATION	1. Compiler Assembly Language File (RA) 2. Assembler Relocatable Binary File (RL); Optional Listing File (LS)
Relocatable Binary File (RL)	RELOCATION	Loader Image File (LD); Optional Loader Map (MP)
Loader Image File (LD)	EXECUTION	Executed Program

Note that the compilation is a two-step process where the source program, after being compiled, is automatically assembled by the system. Once a program has been written and debugged, it may be stored as a loader image file and executed whenever required without the need for further compilation or relocation.

7.2.1 FORTRAN IV Compiler

The FORTRAN IV compiler accepts one FORTRAN source language program or subroutine as input. It then examines each FORTRAN statement for validity, and produces a list of error diagnostics plus a relocatable binary version of the source program, along with an optional annotated source listing, as output. A job containing one or more subroutines may be run by compiling and assembling the main program and each subroutine separately, then combining them with the loader. When compilation is completed, the compiler automatically chains to the postprocessor.

An internal statement number (ISN) is assigned to each FORTRAN IV statement sequentially, in octal notation, beginning with ISN 2 at the first FORTRAN statement. When an error is encountered during compilation, the compiler prints a 2-character error code, followed by the ISN of the offending statement, on the console terminal during pass 2. When a listing is produced, an extended error message is printed below every erroneous statement in the listing. Certain errors cause an immediate return to the Monitor, thereby preventing the printing of a listing file. Table 7-4 lists the FORTRAN compiler error messages and describes the error condition indicated by each message.

The RALF postprocessor is called automatically to assemble the output of the compiler. The postprocessor reads the special assembly language file produced by the compiler as input, generates a binary relocatable file, and routes it to the first output file specified to the compiler. If this file has a null extension, the default extension .RL is supplied.

7.2.1.1 Using the COMPILE Command – Compilation of a source program is done by typing a COMPILE command. This command has the following format:

```
.COMPILE DEV:RALF.RL,DEV:LIST.LS,MAP.MP<INFIL.FT(Options)
```

where

DEV:RALF.RL	is the relocatable binary RALF module;
DEV:LIST.LS	is the annotated listing of the compiler output;
MAP.MP	is the loader symbol map:

<(left-angle bracket)	separates output file specification from input file specification
INFIL	is the input file; and
Options	is any string of alphabetic characters that designates run-time options desired.

The parenthesis may be omitted if each run-time option specification character is preceded with a slash (/). If you enter the first output file name with a null extension, the compiler appends the default extension .RL. If the second output file is a directory device file with a null extension, the assembler appends the default extension .LS. If no listing or map file is specified, none will be generated. Omitting all output specifications causes the compiler to generate a relocatable binary file that has the same name as the input file but with an .RL extension.

The compiler accepts the run-time option specifications (Table 7-3), any combination of which may be requested by entering the appropriate alphabetic character(s) in the command string. Any options recognized by the loader or the run-time system may be entered along with the compiler options. The options are passed to the loader automatically unless chaining is suppressed (by an error condition or omission of the /L option specification), in which case they are ignored.

Table 7-3 FORTRAN IV Compiler Run-Time Options

Option	Operation
/N	Suppress compilation of ISNs. This option reduces program memory requirements by two words per executable statement; however, it also prevents full error traceback at run time.
/Q	Optimize cross-statement subscripting during compilation. This option should not be requested when any variable that appears in a subscript is modified either by referencing a variable equivalent to it or via a SUBROUTINE or FUNCTION call (whether as an argument or through COMMON):
/G	Chain to the loader when assembly is complete and chain to the run-time system following creation of a loader image file (equivalent to the EXECUTE command).
/L	Chain to the loader when assembly is complete. If the /L option is not specified, the system will return to the Monitor upon completion.

7.2.1.2 Examples of Compilation – The following examples illustrate the use of the various options with the COMPILE command:

.COMPILE PROG	Compiles PROG.FT and produces RALF module PROG.RL on DSK:
.COMPILE PROG/G	Compiles PROG.FT into PROG.RL, links it into PROG.LD, then loads it into memory and executes it. No listing files are produced. This command is the same as EXECUTE PROG.
.COMPILE PROG,LIST<DSK:PROG	Produces PROG.RL and a compiler listing file called LIST.LS.

7.2.1.3 Compiler Error Messages – During compilation pass 2, error messages are displayed at the terminal as a 2-character error message followed by the ISN of the erroneous statement. Typing CTRL/O at the terminal suppresses the printing of error messages. If a listing was requested, an extended error message is appended to the listing, immediately following the erroneous statement, during pass 3. Except where indicated in Table 7-4, errors located by the compiler do not halt processing.

Table 7-4 Compiler Error Messages

Error Code	Meaning
AA	More than six subroutine arguments are arrays.
AS	Bad ASSIGN statement.
BD	Bad dimensions (too big or syntax) in DIMENSION, COMMON or type declaration.
BS	Illegal in BLOCK DATA Program.
CL	Bad COMPLEX literal.
CO	Syntax error in COMMON statement.
DA	Bad syntax in DATA statement.
DE	This type of statement illegal as end of DO loop (that is, GO TO, another DO).
DF	Bad DEFINE FILE statement.
DH	Hollerith field error in DATA statement.
DL	DATA list and variable list are not same length.
DN	DO-end missing or incorrectly nested. This message is not printed during pass 3. If it is followed by the statement number of the erroneous statement, rather than the ISN.
DO	Syntax error in DO or implied DO.
DP	DO loop parameter not integer or real.
EX	Syntax error in EXTERNAL statement.
GT	Syntax error in GO TO statement.
GV	Assigned or computed GO TO variable must be integer or real.
HO	Hollerith field error.
IE	Error reading input file. Control returns to the Keyboard Monitor.
IF	Logical IF statement cannot be used with DO, DATA, INTEGER, etc.
LI	Argument of logical IF is not type Logical.
LT	Input line too long, too many continuations.
MK	Misspelled keyword.
ML	Multiply defined line number.
MM	Mismatched parenthesis.
MO	Expected operand is missing.
MT	Mixed variable types (other than integer and real)
OF	Error writing output file. Control returns to the Keyboard Monitor.
OP	Illegal operator.
OT	Type / operator use illegal (for example, A.AND.B where A and/or B not typed Logical).
PD	Compiler stack overflow; statement too big and/or too many nested loops.
PH	Bad program header line.
QL	Nesting error in EQUIVALENCE statement.
QS	Syntax error in EQUIVALENCE statement.
RD	Attempt to redefine the dimensions of a variable.
RT	Attempt to redefine the type of a variable.
RW	Syntax error in READ/WRITE statement.
SF	Bad arithmetic statement function.
SN	Illegal subroutine name in CALL.
SS	Error in subscript expression, that is, wrong number, syntax.
ST	Compiler symbol table full, program too big. Causes an immediate return to the Monitor.
SY	System error, that is, PASS2O.SV or PASS2.SV missing, or no room on system for output file. Causes an immediate return to the Monitor.
TD	Bad syntax in type declaration statement.

Continued on next page

Table 7-4 (Cont.) Compiler Error Messages

Error Code	Meaning
US	Undefined statement number. This message is not printed during pass 3. It is followed by the statement number of the erroneous statement, rather than the ISN.
VE	Version error. One of the compiler programs is absent from SYS or is present in the wrong version.

7.2.2 FORTRAN IV Loader

The FORTRAN IV loader accepts up to 128 relocatable binary modules as input and links the modules, along with any necessary library components, to form a loader image file that may be loaded and executed by the run-time system. Once all relocatable binary modules and library components have been assigned to some portion of memory and linked, absolute addresses are assigned to the relocatable binary text.

If the /L or /G option is specified with the COMPILE command, the loader is called automatically to relocate the output of a successful compilation.

The loader must be called separately to link a set of previously compiled relocatable binary modules by using the LOAD command.

When several files or subprograms are loaded, the format of the LOAD command is as follows:

```
.LOAD DEV:OUTFIL.LD,MAP.LS<DSK:INFIL1.RL,...,INFIL9.RL (options)
```

When programs are individually loaded the format is as follows:

```
.LOAD INFIL.RL
```

OUTFIL.LD is the loader image output file and MAP.LS is the loader symbol map output file. The input files may be either relocatable binary modules or a library file, and "Options" is a string of alphabetic characters that designates any run-time options desired.

If the output file is not specified, the input file is loaded and given the default extension .LD by the system. The loader symbol map is routed to the second output file provided that a second file is specifically defined. If this is a directory device file with no extension, the default extension .LS is assumed.

The system can only accept nine input file specifications for each command line. If additional files must be input, the /C (continue) option is used, since it permits the additional files to be put on the following line. Other Loader run-time options are given in Table 7-5.

A LOAD command is usually terminated by pressing the RETURN key. If a LOAD command is terminated by the ESCape key, the Command Decoder, a system program, will be called (see Appendix D for a description of the Command Decoder). The Command Decoder indicates that it is running by prompting with an asterisk (*) on the terminal screen. The function of this program is to allow the system to accept file I/O specifications. Pressing the ESCape key again executes the command line. This feature is discussed in Section 7.2.3.

Any error recognized by the loader during generation of a loader image file results in an error message, immediately following the input specification line that caused the error condition. Table 7-6 lists the loader error messages and describes the error condition indicated by each message.

Table 7-5 Loader Run-Time Options

Option	Operation
/C	Continue the current line of input on the next line of input. When specifying input files to the loader, there may be more than nine files in a command line. The C option permits the additional files to be put on the following line.
/G	Treat the current line as the last line of input, and chain to the FORTRAN IV run-time system when finished.
/S	Include system symbols in the loader symbol map. System symbols are identified by an initial number sign (#). This option is only valid when a symbol map output file was specifically defined.

7.2.2.1 Examples of Loading – The following examples illustrate the use of the various options with the LOAD command:

.LOAD POWER.RL

Loads POWER.RL and produces a loader image file POWER.LD.

.LOAD RXA1:POWER.RL/G

Loads POWER.RL, produces a loader image file POWER.LD, chains to the run-time system, and executes the program.

.LOAD PROG.LD,RXA1:MAP<MAIN.RL,SUBA.RL

Loads and links MAIN.RL and SUBA.RL to produce PROG.LD, and produces MAP.LS, a loader symbol map output file, on diskette RXA1. Using the type command .TYPE RXA1:MAP will display the MAP file on the screen. Typing .EXE PROG.LD will execute the program.

.LOAD PROG.LD<PROG.RL,PROG1.RL (ESC)*(more file specifications) (ESC)

Loads and links two input files to produce PROG.LD, then calls a special system program for accepting file I/O specifications (See Section 7.2.3). The second ESCape executes the entire command.

7.2.2.2 Loader Error Messages – The loader displays error messages on the terminal during generation of a loader image file. Except where indicated in Table 7-6, loader errors are fatal. The loader returns to the Monitor when a fatal error condition is encountered.

Table 7-6 Loader Error Messages

Error Message	Meaning
BAD INPUT FILE	An input file was not a relocatable binary module.
BAD OUTPUT DEVICE	The loader image file device was not a directory device, or the symbol map file device was a read-only device. The entire line is ignored.
MIXED INPUT	The L option was specified on a line that contained some file other than a library file. The library file (if any) is accepted. Any other input file specification is ignored.
NO MAIN	No relocatable binary module contained the section #MAIN.
OVER CORE	The loader image requires more than 16K of memory.
OVER IMAG	Output file overflow in the loader image file.
OVER SYMB	Symbol table overflow. More than 253 (decimal) symbols in one FORTRAN job.
TOO MANY RALF FILES	More than 128 input files were specified.
EX	The symbol is referenced but not defined.
ME	Multiple Entry. The symbol is multiply defined.

7.2.3 FORTRAN IV Run-Time System (FRTS)

The run-time system reads, loads, and executes a loader image file produced by the loader. It evaluates arithmetic and logical operations. It also configures a software I/O interface between the FORTRAN IV program and the OS/78 operating system, and then monitors program execution to direct I/O processes and identifies certain types of run-time errors. The run-time system is automatically called to load and execute the loader image file produced by the loader whenever the /C option is specified to the loader. When chained to the loader, the run-time system reacts in one of two ways. If the LOAD command line was terminated by pressing the RETURN key, the program is executed. If the LOAD command line was terminated with an ESCape, a system program called the Command Decoder is called and indicates that it is running by printing an asterisk (*).

In response to the asterisk, file specifications can be input to the run-time system. This allows a source program to be written that refers to I/O devices as integer costants or variables. Such a program may be compiled, assembled, and loaded into an image file. This image file may be run any number of times, each time specifying different actual peripheral devices. Thus logical unit 8 may refer in one run to the console terminal and in another run to a diskette file.

Of the nine I/O unit numbers available under FORTRAN IV, two are initially assigned to FORTRAN internal device handlers by the system:

I/O Unit	Internal Handler	Comments
3	Line printer	LA78 only.
4	Console terminal	Double buffered output, single character input.

The FORTRAN internal handlers listed above are not the same as the OS/78 device handlers. The FORTRAN internal handlers are designed for ASCII text only and will not transfer noncharacter data.

Additional unit numbers may be assigned, in addition to those listed above, to the FORTRAN internal device handlers by typing (in response to the asterisk generated by the Command Decoder).

/n=m

where

n	is a different unit number (1 to 9) that is also to be assigned to that internal handler; and
m	is the I/O unit number (3 or 4) of the internal handlers.

This specification causes all program references to logical unit n to perform I/O to device m. For example,

/6=3	Assigns the FORTRAN internal line printer handler as I/O unit number 6, in addition to unit number 3.
/3=4	Assigns I/O unit number 3 to the FORTRAN console terminal handler instead of the internal line printer handler.

I/O unit numbers may be assigned to OS/78 device handlers for nondirectory devices by typing (in response to the asterisk generated by the Command Decoder).

DEV:/n

where

DEV:	is the standard or assigned designation for any supported nondirectory device; and
n	is an I/O unit number (1 to 9).

For example,

LQP:/3	Specifies the OS/78 LQP line printer handler to be used as device #3 instead of the FORTRAN internal line printer handler.

Existing directory device files may be assigned I/O unit numbers by typing (in response to the asterisk generated by the Command Decoder)

DEV:FILE.EX/n

where

DEV:FILE.EX	is the standard OS/78 designation for an existing directory device file; and
n	is an I/O unit number (1 to 9).

For example,

RXA1:FORIO.TM/4	Assigns unit number 4 to Diskette file FORIO.TM rather than to the FORTRAN internal console terminal handler, where FORIO.TM is an existing file on Diskette unit 1.

A directory device file that does not presently exist may be assigned a FORTRAN I/O unit number in the same manner by entering it as an output file on the specification line; however, only one such file may be created on any particular device. For example:

FORIO.TM</9	Assigns unit number 9 to file DSK:FORIO.TM, which has not been created at load time.

In any case, only one device or file specification is permitted on each line, and no more than six directory device files may be created by the FORTRAN program. Excess files after the sixth are accepted and written, but they will not be closed. If a file created by the program has the same file name and extension as a pre-existing file, the old file is automatically deleted when the new file is closed.

The Command Decoder "[n]" specification may be used to optimize storage allocation when assigning files that do not yet exist, where n is a decimal number that indicates the maximum expected length of the file, in blocks.

Each time a run-time I/O specification is terminated by pressing the RETURN key, the Command Decoder is recalled to accept another specification. When a specification is terminated by the ESCape key, the program is run.

The following examples illustrate the use of device and file specifications.

Example 1:

```
C       WRITE.FT PROGRAM
        DIMENSION ILT(200)
        INTEGER ILT
C       SPECIFY LOGICAL UNIT NUMBER FOR TTY AS 2 INSTEAD OF 4
1       WRITE (2,10)
10      FORMAT (1X,'WRITING AND READING ASCII SEQ. DATA FILE')
        DO 3 I=1,200
3       ILT(I)=I**2
        WRITE (8,20) (ILT(I),I=1,200)
20      FORMAT (1H,//,20(10I7,/))
        REWIND 8
C       NOW READ DATA FILE DATA.DA FROM MASS STORAGE DEVICE
        READ (8,20) ILT
C       OUTPUT FILE TO TTY
        WRITE (4,20) ILT
        END
```

The above program raises 200 sequential numbers to the power of two. The following sequence of specifications are typed using the COMPILE command as follows:

```
.COMPILE WRITE.FT/G (ESC) */2=4
*RXA1:DATA.DA</8 (ESC)
```

The /G option in the command line will call the run-time system to execute the program. Pressing the ESCape key after the command calls the Command Decoder, allowing device and file specifications to be declared. In this case, the terminal (4) is assigned to unit number 2 while the load image file is assigned number 8. A file DATA.DA is created on diskette 1 that contains the results of the program. The contents of this file are then displayed on the terminal screen. The command and the specification strings are executed by the second ESCape.

Example 2:

The second example shows an output file RAY.DA being created on the diskette by PROG1.FT, and then being read from the diskette by PROG2.FT.

```
C       THIS PROGRAM WRITES A RECORD OF 400 VARIABLES
C       INTO A FILE CALLED RAY.DA
        DIMENSION RAY(400)
        INTEGER RAY
        DEFINE FILE 1 (1,400,U,J)
        J=1
        DO 5 I=1,400
5       RAY(I)=2*I
        WRITE (1'J) (RAY(I),I=1,400)
        CALL EXIT
        END
```

The above program writes a record into file RAY.DA. The following command and specification strings are typed to accomplish this.

```
.COMPILE PROG1.FT/G (ESC) *RAY.DA</1 (ESC)
```

```
C       READ DIRECT ACCESS FILE RAY.DA FROM MASS STORAGE
C       DEVICE CREATED BY PREVIOUS PROGRAM
        DIMENSION RAY(400)
        INTEGER RAY DEFINE FILE 1(1,400,U,J)
        J=1
        READ (1'J) (RAY(I),I=1,400)
100     FORMAT (1H,//,40(10I6,/))
C       DUMP CONTENTS OF DATA FILE ONTO TTY
        WRITE (4,100) RAY
        CALL EXIT
        END
```

The above program then reads out the results of PROG1.FT, and displays the contents of file RAY.DA on the terminal screen. This is done by typing the following command line.

```
.COMPILE PROG2.FT/G (ESC) *RAY.DA/1 (ESC)
```

Although existing files are specified as though they were input files and nonexistent files are specified as though they were output files, any file that has been assigned a unit number may be used for either input or output. Run-time system option specifications are described in Table 7-7.

The run-time system recognizes two classes of error conditions. Certain errors are diagnosed while the load image file is being read from a storage device and loaded into memory. Other errors may occur during execution of the FORTRAN program. Both classes of run-time errors are identified on the console terminal. Table 7-8 lists the FRTS error messages and describes the error condition indicated by each message. The run-time system error traceback feature provides automatic printout of statement numbers ISNs corresponding to the sequence of executable statements that terminated in an error condition. At least one statement number is always printed. This number identifies the erroneous statement or, in certain cases, the last correct statement executed prior to the error. When a statement was compiled under the /N option, however, the system cannot generate

Table 7-7 Run-Time System Options

<table>
<tr><th>Option</th><th>Operation</th></tr>
<tr><td>/C</td><td>Carriage control switch. The first character on every output line is processed as a carriage control character by all FORTRAN internal handlers and also by the OS/78 handlers TTY and LPT. The first character on every output line is processed as data, in the same manner as any other character, by all OS/78 handlers except TTY and LPT. Entering a C option specification on the command line that assigns an I/O unit number to a particular handler reverses the processing of carriage control characters for that device. Thus,

TEMP(2C)

assigns file DSK:TEMP. as I/O unit 2. The /C option causes the first character of every output line to be processed as a carriage control character. If C were not specified, these characters would be processed as data.

/C/6=3

assigns the FORTRAN internal line printer handler as I/O unit 6, as well as unit 3. The first character of every line will be processed as a carriage control character on unit 3, and as a character of data on unit 6.</td></tr>
<tr><td>/E</td><td>Ignore the following run-time system errors, any of which indicates that an error was detected earlier in the compilation loading process:

1. Illegal subroutine call.
2. Reference to an undefined symbol.</td></tr>
</table>

meaningful statement numbers during traceback. When a statement is reached through any form of GOTO, the line number for error traceback is not reset. Thus, an error in such a line will give the number of the last executed line in the error traceback.

The console terminal serves as FORTRAN I/O unit 4 for both input and output. Terminal input is automatically echoed on the console screen. In addition, the run-time system monitors the keyboard continually during execution of a FORTRAN program. Typing CTRL/C at any time causes an immediate return to the Monitor. Typing CTRL/B branches to the system traceback routine, and then exits to the monitor. This traceback routine generates a printout, similar to the error traceback, including the current subroutine, the line number in the next higher level subroutine from which it was called, and so forth, to the main program. This facilitates locating infinite loops when debugging a program. The following additional special characters are recognized by the console terminal handler and processed as shown:

DELETE	Deletes last character accepted.
CTRL/U	Deletes current line of input.
CTRL/Z	Signals end-of-file on input.

Tentative output files (that is, files created by the FORTRAN program) are closed automatically upon successful completion of program execution provided that one of the following conditions occurs:

1. An END FILE statement referencing the file was executed. FRTS assigns a file length equal to the actual length of the file.
2. The last operation performed on the file was a write operation. FRTS proceeds as though an END FILE statement had been executed.
3. A DEFINE FILE statement referencing the file was executed but an END FILE statement was not executed. Upon completion of program execution, FRTS assigns a file length equal to the length specified in the DEFINE FILE statement.

Execution of a REWIND statement does not close a tentative file, nor does it modify the tentative file length.

7.2.3.1 Run-Time System Error Messages – The run-time system generates two classes of error messages. Messages listed in Table 7-8 identify errors that may occur during execution of a FORTRAN program and errors that may be encountered when the run-time system is reading a loader image file into memory in preparation for execution, or accepting I/O unit specifications. Except where indicated, all run-time system errors cause full traceback and an immediate return to the monitor. Nonfatal errors cause partial traceback, sufficient to locate the error, and execution continues.

Table 7-8 Run-Time System Error Messages

Error Message	Meaning
BAD ARG	Illegal argument to library function.
CAN'T READ IT!	I/O error on reading loader image file.
D.F. TOO BIG	Random access file requirements exceed available storage.
DIVIDE BY 0	Attempt to divide by zero. The resulting quotient is set to zero and execution continues.
EOF ERROR	End of file encountered on input.
FILE ERROR	Any of the following conditions occurred: 1. A file specified as an existing file was not found. 2. A file specified as a nonexistent file would not fit on the designated device. 3. More than one nonexistent file was specified on a single device. 4. The file specification contained an asterisk (*) as name or extension.
FILE OVERFLOW	Attempt to write outside file boundaries.
FORMAT ERROR	Illegal syntax in FORMAT statement.
INPUT ERROR	Illegal character received as input.
I/O ERROR	Error reading or writing a file, tried to read from an output device, or tried to write on an input device.

Continued on next page

Table 7-8 (Cont.) Run-Time System Error Messages

Error Messages	Meaning
MORE CORE REQUIRED	The space required for the program, the I/O device handlers, the I/O buffers, and the resident Monitor exceeds the available memory.
NO DEFINE FILE	Direct access I/O attempted without a DEFINE FILE statement.
NO NUMERIC SWITCH	The referenced FORTRAN I/O unit was not specified to the run-time system.
NOT A LOADER IMAGE	The first input file specified to the run-time system was not a loader image file.
OVERFLOW	Result of a computation exceeds upper bound for that class of variable. The result is set equal to zero and execution continues.
PARENS TOO DEEP	Parentheses nested too deeply in FORMAT statement.
SYSTEM DEVICE ERROR	I/O failure on the system device.
TOO MANY HANDLERS	Too many I/O device handlers are resident in memory, or files have been defined on too many devices.
USER ERROR	Illegal subroutine call, or call to undefined subroutine. Execution continues only if the E option was requested.
UNIT ERROR	I/O unit not assigned, or incapable of executing the requested operation.

7.2.4 FORTRAN IV Library: Library Functions and Subroutines

The OS/78 FORTRAN IV system contains a general purpose FORTRAN library named FORLIB.RL. FORLIB.RL contains mathematical functions and miscellaneous subroutines.

Library functions and subroutines are called in the same manner as user written functions and subroutines. This section lists the library components that are available to FORTRAN programs and illustrates calling sequences, where necessary. Arguments must be of the correct number and type, but need not have the same name as those shown in the illustrative examples. Certain library routines are used by the FORTRAN system programs and are not available to a user's FORTRAN program. These routines may be identified by the initial number sign (#) in the entry point or section name, and are not included in the following:

ABS (ABSOLUTE VALUE)

ABS calculates the absolute value of a real variable by leaving the variable unchanged if it is positive (or zero) and negating the variable if it is a negative.

ACOS (ARC-COSINE FUNCTION)

ACOS calculates and returns the primary arc-cosine (in radians) or a real argument in the range [-1,1] according to the following relation:

$$\text{IF } x>0.0,\ \text{ACOS}(x)=\text{ATAN}\left[\frac{\text{SQRT}(1-X^2)}{X}\right]$$

$$\text{IF } x<0.0,\ \text{ACOS}(x)=\pi+\text{ATAN}\left[\frac{\text{SQRT}(1-X^2)}{X}\right]$$

$$\text{IF } x=0.0,\ \text{ACOS}(x)=\pi/2.0$$

AINT (REAL TO INTEGER)

AINT is a truncation function. Given a real argument, it truncates the fractional part of the argument and returns the integral part as an integer. This is accomplished by taking the absolute value of the argument, deleting the fractional portion, and then restoring the original sign. AINT, IFIX, and INT perform identical functions.

ALOG (NATURAL LOGARITHM)

ALOG calculates and returns the natural (Naperian) logarithm of a real argument greater than zero. Any negative or zero argument returns an error message and a value of 0.0. The algorithm used is an 8-term Taylor series approximation.

ALOG10 (COMMON LOGARITHM)

ALOG10 calculates and returns the common (base 10) logarithm of a real argument greater than zero. Any negative or zero argument returns an error message and a value of 0.0. The calculation is accomplished by calling ALOG to compute the natural logarithm and executing a change of base.

AMAX0 (MAXIMUM VALUE)

AMAX0 accepts an arbitrary number of integer arguments and returns a real value equal to the largest of the arguments.

AMAX1 (MAXIMUM VALUE)

AMAX1 accepts an arbitrary number of real arguments and returns a real value equal to the largest of the arguments.

AMIN0 (MINIMUM VALUE)

AMIN0 accepts n arbitrary number of integer arguments and returns a real value equal to the smallest of the arguments.

AMIN1 (MINIMUM VALUE)

AMIN1 accepts an arbitrary number of real arguments and returns a real value equal to the smallest of the arguments.

AMOD (A MODULO B)

AMOD accepts two real arguments and returns a real value equal to the remainder when the first argument is divided by the second argument. If the second argument is not large enought to prevent overflow, an error message and a value of 0.0 are returned.

ASIN (ARC-SINE)

ASIN calculates and returns the primary arc-sine (in radians) of a real argument in the range [-1, 1] according to the relation

ASIN(X)=ATAN(X/SQRT(1-X**2))

If the argument falls outside the range [-1, 1], an error message results.

ATAN (ARC-TANGENT)

ATAN calculates and returns the primary arc-tangent (in radians) of a real argument. The argument is first reduced according to the following relations:

(1) If $x<2^{-14}$,	atan(x)=x
(2) If $x>2^{-14}$,	atan(x)=1/x
(3) If x>1.0,	atan(x)=π/2-atan(1/x)
(4) If x<0,	atan(x)=-atan(-x)

and the arc-tangent is then computed by a power series approximation.

ATAN2 (ARC-TANGENT OF TWO ARGUMENTS)

ATAN2 accepts two real arguments, assumed to be an abscissa and an ordinate, and calculates the arc-tangent of the quotient of the first argument divided by the second argument. This is accomplished by calling ATAN to find the principal arc-tangent of the quotient and then adjusting the result, depending on the quadrant in which a point defined by the arguments falls, according to the following relations:

argument in first quadrant	atan2(y,x)=atan(y/x)
argument in second quadrant	atan2(y,x)=atan(y/x)-π
argument in third quadrant	atan2(y,x)=atan(y/x)-π
argument in fourth quadrant	atan2(y,x)=atan(y/x)+π

CGET (CHARACTER GET SUBROUTINE)

The calling sequence

CALL CGET (STRING,N,CHAR)

causes the Nth character to be unpacked from STRING and stored in CHAR as a variable in the range [0,63], where STRING is a character string in A6 format.

CHKEOF (CHECK FOR END-OF-FILE SUBROUTINE)

CHKEOF accepts one real, integer or logical argument. After the next formatted read operation, this argument will be set to nonzero if the logical end-of-file was encountered, or to 0 if the logical end-of-file was not encountered. The following example illustrates the use of CHKEOF:

```
.
.
.
CALL CHKEOF(EOF)
READ (N,101)DATA
IF (EOF.NE.0) GO TO 999
.
.
.
```

CLOCK (INITIALIZE CLOCK SUBROUTINE)

The purpose of the CLOCK subroutine is to initialize the KK8-B real-time clock. The calling sequence for the subroutine is as follows:

```
CALL CLOCK (functn,rate)
```

where function can have the value 0 or 8, specifying multiple or single A/D channel input, respectively; the value of rate is preset to 100 (specifying 100 Hz). Any value of functn other than 0 or 8 causes an error message; any value of rate other than 100 is ignored.

COS (COSINE FUNCTION)

COS calculates and returns the cosine of a real argument (in radians) by applying the identity:

```
COS(X)=SIN(X+ /2)
```

COSH (HYPERBOLIC COSINE FUNCTION)

COSH calculates and returns the hyperbolic cosine of a real argument according to the relations:

If $|x| \leqslant 88.029$

$$COSH(x) = 1/2\left(EXP(x) + \frac{1.0}{EXP(x)}\right)$$

If $|x| > 88.028$ and $|x| - \log_e 2 \leqslant 88.028$

$$COSH(x) = EXP(|x| - \log_e 2)$$

If $|x| - \log_e 2 > 88.028$

$$COSH(x) = 377737777777_8$$

and an error message is returned.

CPUT (CHARACTER PUT SUBROUTINE)

The calling sequence

```
CALL CPUT (STRING,N,CHAR)
```

causes CPUT to insert CHAR as the Nth character in STRING, where STRING is a character string stored in A6 format, and CHAR is a number in the range [0,63] which is interpreted as a character. The following program illustrates the use of CGET and CPUT.

```
      DATA STR/'HEY!'/
      WRITE(4,100) STR
100   FORMAT ('    HEY! IN ASCII    ',A6)
      WRITE(4,101)
101   FORMAT('    HEY! IN DECIMAL')
      DO 10 I=1,4
      CALL CGET(STR,I,ICHAR)
      WRITE(4,102) ICHAR
 10   CONTINUE
102   FORMAT(I6)
      DO 20 I=1,6
      J=2*I
      CALL CPUT(STR,I,J)
 20   CONTINUE
      WRITE(4,103) STR
103   FORMAT('    NEW STRING    ',A6)
      CALL EXIT
      END

.COMPILE TCHRC/G
  HEY! IN ASCII  HEY!
  HEY! IN DECIMAL
     8
     5
    25
    33
  NEW STRING    BDFHJL
```

DATE (DATE SUBROUTINE)

DATE accepts three integer arguments, accesses the current system date, and returns an integer from 1 to 12 corresponding to the current month as the first argument, an integer from 1 to 31 corresponding to the current day as the second argument, and an integer from 1970 to 1999 corresponding to the current year as the third argument.

DIM (POSITIVE REAL DIFFERENCE)

DIM calculates and returns the positive difference of two real arguments. If the first argument is larger than the second argument, DIM returns the difference between the arguments; if the first argument is less than or equal to the second argument, DIM returns 0.0.

EXP (EXPONENTIAL FUNCTION)

EXP calculates and returns the exponential function of a real argument. The algorithm uses a numerical method after Kogbetliantz (IBM Journal of Research and Development, April, 1957, pp 110-5).

EXP3 (BASE RAISED TO AN EXPONENT)

EXP3 accepts two real or integer arguments, that is, a base and an exponent and performs the following calculation:

$$a=b^e$$

FLOAT (INTEGER-TO-FLOATING-POINT CONVERSION)

FLOAT accepts an integer argument and returns a real variable equal to the argument.

IABS (INTEGER ABSOLUTE VALUE FUNCTION)

IABS calculates and returns the absolute value of an integer variable by leaving the variable unchanged if it is positive (or zero), and negating the variable if it is negative.

IDIM (INTEGER POSITIVE DIFFERENCE FUNCTION)

IDIM calculates and returns the positive difference of two integer arguments. If the first argument is larger than the second argument, IDIM returns the difference between the arguments; if the first argument is less than or equal to the second argument, IDIM returns a value of 0.

IFIX (FLOATING-POINT-TO-INTEGER FUNCTION)

IFIX is a floating-point truncation function. Given a real argument, it truncates the fractional part of the argument and returns the integral part as an integer. IFIX, AINT, and INT perform the same function.

INT (FLOATING-POINT-TO-INTEGER)

INT is a floating-point truncation function that performs the same function as AINT and IFIX.

ISIGN (INTEGER TRANSFER OF SIGN FUNCTION)

ISIGN accepts two integer arguments, calculates the absolute value of the first argument, and returns this value if the second argument is positive (or zero), or the negative of this value if the second argument is negative.

MAX0 (MAXIMUM VALUE)

MAX0 accepts an arbitrary number of integer arguments and returns an integer result equal to the largest of the arguments.

MAX1 (MAXIMUM VALUE)

MAX1 accepts an arbitrary number of real arguments and returns an integer result equal to the largest of the arguments.

MIN0 (MINIMUM VALUE FUNCTION)

MIN0 accepts an arbitrary number of integer arguments and returns an integer vaue equal to the smallest of the arguments.

MIN1 (MINIMUM VALUE FUNCTION)

MIN1 accepts an arbitrary number of real arguments and returns an integer value equal to the smallest of the arguments.

MOD (INTEGER A MODULO B FUNCTION)

MOD accepts two integer arguments and returns an integer value equal to the remainder when the first argument is divided by the second argument. If the second argument is not large enough to prevent overflow, an error message and a value of zero are returned.

SIGN (TRANSFER OF SIGN)

SIGN accepts two real arguments, calculates the absolute value of the first argument, and returns this value if the second argument is positive (or zero), or the negative of this value if the second argument is negative.

SIN (SINE FUNCTION)

SIN calculates and returns the sine of a real argument (in radians). The argument is reduced to the first quadrant, and the sine is then computed from a Taylor series expansion.

SINH (HYPERBOLIC SIGN)

SINH calculates and returns the hyperbolic sine of a real argument according to the following relations:

If $0.10<|x|<87.929$, $SINH(x) = 1/2[EXP(x) - \frac{1}{EXP(x)}]$

If $|x|\leqslant 0.10$, $SINH(x) = x+x^{3/6}+ x^{5/120}$

If $|x|>88.028$, $SINH(x) = [EXP(|x| - \log_e 2)] \cdot [signum(x)]$

SQRT (SQUARE ROOT FUNCTION)

SQRT calculates and returns the (positive) square root of a positive real argument. Any negative argument results in an error message.

TAN (TANGENT FUNCTION)

TAN calculates and returns the tangent of a real argument (in radians). This is accomplished by computing the quotient of the sine of the argument divided by the cosine of the argument; thus, if the cosine of the argument is zero, an error message is returned.

TANH (HYPERBOLIC TANGENT)

TANH calculates and returns the hyperbolic tangent of a real argument by computing the quotient of the hyperbolic sine of the argument divided by the hyperbolic cosine of the argument.

TIME (READ TIME OF DAY)

TIME may be called as a subroutine with one real or integer argument, or as a function with a dummy argument. It returns the elapsed time since the clock was started. This result will be in seconds.

7.3 RUNNING OS/78 FORTRAN IV PROGRAMS

The first example that will be used is the power of two program, POWER.FT, that was created with the Editor, and described in Chapter 4. It is repeated here for convenience.

```
C     FORTRAN DEMONSTRATION
C     COMPUTE AND PRINT POWERS OF TWO
      DIMENSION A(16)
      WRITE(4,15)
15    FORMAT(1H ,'POWER OF TWO')
      DO  20 N=1,16
      A(N)=2.**N
20    CONTINUE
      WRITE (4,25) (N,A(N),N=1,16)
25    FORMAT(1H ,'2** ',I2,'=',F10.1)
      STOP
      END
```

7.3.1 Executing a Program

This program is executed by typing

```
.EXE POWER.FT
```

This command compiles, loads, and executes the program with the following results displayed on the screen:

```
 POWER OF  TWO
2**   1=          2.0
2**   2=          4.0
2**   3=          8.0
2**   4=         16.0
2**   5=         32.0
2**   6=         64.0
2**   7=        128.0
2**   8=        256.0
2**   9=        512.0
2**  10=       1024.0
2**  11=       2048.0
2**  12=       4096.0
2**  13=       8192.0
2**  14=      16384.0
2**  15=      32768.0
2**  16=      65536.0
```

7.3.2 Compiling a Program

Other commands can be used to compile and load the program in sequential steps, and they are required when subroutines are used.

The first command is the COMPILE command. This command compiles the source program POWER.FT and generates a relocatable binary file. The format of the command is

```
.COMPILE POWER.FT
```

After execution of this command, the compiled file is listed in the directory as POWER.RL. A source listing of the compiled file can be output to the terminal by typing

```
.COMPILE POWER.RL,TTY:<POWER.FT
```

This command line will display the following annotated listing on the terminal screen:

```
  FORTRAN IV        4AAAA             17-JUN-77            PAGE  ONE

          C         FORTRAN DEMONSTRATION
          C         COMPUTE AND PRINT POWERS OF TWO
0002                DIMENSION A(16)
0003                WRITE (4,15)
0004      15        FORMAT(1H ,'POWER OF TWO')
0005                DO 20 N=1,16
0006                A(N)=2.**N
0007      20        CONTINUE
0010                WRITE (4,25) (N,A(N),N=1,16)
0011      25        FORMAT(1H ,'2** ',I2,'=',F10.1)
0012                STOP
0013                END
```

This file is useful in tracking any errors that may have been made in the program.

The compiler options discussed previously may be specified when using the COMPILE command. For example, if the /G option was invoked in the command line

```
.COMPILE POWER.FT/G
```

the program would be compiled, and then chained to the loader to create an image file, and finally executed by being chained to the run-time system.

If the /L option is specified with the COMPILE command, the program will be chained to the loader, creating a POWER.LD file, but will not be executed.

7.3.3 Creating a Loader Image File

After compilation, use the LOAD command to create a loader image file by typing

```
.LOAD POWER.RL
```

Again, the various options previously described for the loader may be used with this command. After loading, type

```
.EXECUTE POWER.LD
```

This command will cause the program to be executed and display the results on the terminal screen. This is the fastest method to use if the program does not have to be recompiled or relinked.

If a loader symbol map is desired, type

```
.LOAD POWER.LD,TTY:<POWER.RL
```

The following map for program POWER will be displayed on the terminal screen:

```
LOADER V24A    17-JUN-77

SYMBOL VALUE LVL  OVLY

ARGERR 00204  0   00
EXIT   00223  0   00
#MAIN  10000  0   00
 11000  = 1ST FREE LOCATION


LVL  OVLY LENGTH

 0   00    10601
```

The optional loader symbol map lists all symbols defined in the loader image file. The LVL and OVLY entries apply only to OS/8, a superset of OS/78.

Following the alphabetical list of symbols, the loader prints the address of the first free memory location and the length, in octal words. This information is useful for optimizing memory requirements.

7.3.4 Running Subprograms

The LOAD command is especially useful to link subprograms. Consider the program shown in Figure 7-5, which computes the volume of a regular polyhedron when given the number of faces and the length of one edge. It consists of a main program and a subroutine. The subprogram does the required computation, using a computed GO TO statement to determine whether the polyhedron is a tetrahedron, cube, octahedron, dodecahedron, or icosahedron, then transfers control to the proper arithmetic expression for performing the calculation.

MAIN PROGRAM:

```
C         COMPUTE THE VOLUME OF A REGULAR POLYHEDRON GIVEN
C         THE NUMBER OF FACES AND LENGTH OF ONE EDGE
          COMMON NFACES,EDGE
5         WRITE(4,10)
10        FORMAT(1H ,'TYPE IN NO.OF FACES AND ONE LENGTH EDGE')
          READ(4,20) NFACES,EDGE
20        FORMAT(I2,F8.5)
          CALL SOLVE
          GO TO 5
          STOP
          END
```

SUBPROGRAM SOLVE:

```
C         SUBROUTINE TO SOLVE PROBLEM AND PRINT ANSWER
C         CALLED SOLVE.FT
          SUBROUTINE SOLVE
          COMMON NFACES,EDGE,VOLUME
          IF(NFACES.GT.20)GOTO 8
          CUBED=EDGE**3
          GO TO(6,6,6,1,6,2,6,3,6,6,6,4,6,6,6,6,6,6,6,5,6),NFACES
1         VOLUME=CUBED*0.11785
          GO TO 20
2         VOLUME=CUBED
          GO TO 20
3         VOLUME=CUBED*0.47140
          GO TO 20
4         VOLUME=CUBED*7.66312
          GO TO 20
5         VOLUME=CUBED*2.18170
          GO TO 20
6         WRITE(4,10) NFACES
10        FORMAT(1H0,'NO REGULAR POLYHEDRON HAS',I3,1X,'FACES')
          RETURN
20        WRITE(4,30) VOLUME
30        FORMAT(1H0,'VOLUME=',F10.2)
          RETURN
8         WRITE(4,10)
40        FORMAT (1H0,"DO NOT SPECIFY MORE THAN 20 FACES)
          RETURN
          END
```

Figure 7-5 Main and Subprogram for Calculating the Volume of a Regular Polyhedron

If the number of faces of the solid is other than 4, 6, 8, 12, or 20, or if more than 20 faces are specified the subroutine displays an error message on the terminal. If the correct input parameters are typed in at the terminal keyboard, the calculation is performed, and the answer is displayed on the terminal screen.

When subprograms are used, the main program and the subprograms must be individually compiled. For the example program, both the main program and subprogram are compiled as follows:

```
.COMPILE MAIN.FT
```

and

```
.COMPILE SOLVE.FT
```

which create relocatable binary files.

The modules must now be loaded to make a single image file that can be executed. This is done as follows:

```
.LOAD POLY.LD<MAIN.RL,SOLVE.RL
```

This command creates a file POLY.LD and returns to the Monitor. Execute the program by typing

```
.EXECUTE POLY.LD
```

which results in the program calling for the required input parameters as follows:

```
TYPE IN NO. OF FACES AND ONE LENGTH EDGE
```

Typing a "6" to represent a cube (preceded by a space since the field specification is I2) and "20" as one length edge in accordance with the field specification requirement as follows:

```
b6 20.0
```

results in the answer being displayed on the terminal screen, and a prompt for the next set of input parameters as follows:

```
VOLUME = 8000.00
TYPE IN NO.  OF FACES AND ONE LENGTH EDGE
```

Do not specify blanks or zeroes for the NFACES and EDGES variables. Type CTRL/C to return to the Monitor.

7.3.5 Specifying I/O Devices

A special case of the LOAD command is that in which a system program called the Command Decoder is called up and signifies it is active by printing an asterisk (*). This program allows the system to accept file I/O specifications.

For example, if the POWER.FT program were written with the output unit designation as a 3, that is,

```
WRITE (3,15)
```

the output generated by program execution would be sent to the line printer. However, if your system does not have a line printer available, the output unit designation is changed by typing

```
.LOAD POWER.LD<POWER.RL/G (ESC)
```

Then pressing the ESCape key (which echoes as a dollar sign) calls up the Command Decoder which prompts with an asterisk. Then type /3=4 which assigns I/O unit 3 to the console terminal instead of the line printer. Pressing the ESCape key again executes the program and program output is sent to the terminal. Thus, the command line will appear as follows:

```
.LOAD POWER.LD<POWER.RL/G (ESC) */3=4 (ESC)
```

The Command Decoder program also allows you to store the output of an executed program to a file that has not been created at load time.

For example,

```
.LOAD POWER.LD<POWER.RL/G (ESC) *RXA1:HOLD.TM</4 (ESC)
```

will output the results of the program into a file called HOLD.TM on RXA1. Typing

```
.TYPE RXA1:HOLD.TM
```

will display the contents of this file, that is, the results of the program POWER.FT on the terminal screen.

7.4 FORTRAN IV SOURCE LANGUAGE

The FORTRAN IV language is composed of mathematical-form statements constructed in accordance with precisely formulated rules. FORTRAN (source) language programs consist of meaningful sequences of FORTRAN statements that direct the computer to perform specified operations and computations.

Each line of a FORTRAN source program contains three fields: statement number field, line continuation field, and statement field. A fourth field, the identification field consisting of columns 73 to 80, is ignored by the compiler. It may be used to number statements sequentially or for any other purpose.

A statement number consists of one to five digits entered in columns 1 to 5. Leading zeros or blanks (leading and triling) are retained on the listing, but otherwise ignored in this field. Statement numbers may be assigned in any order, but they must be unique. Any statement referenced by another statement must have a statement number. Statement numbers on specification statements are ignored.

If a FORTRAN statement is so large that it cannot conveniently fit into one statement field, the statement fields of up to five additional lines may be used to specify the complete statement. The first line of a statement must have a blank in column 6. Continuation lines must have some character other than a blank in column 6.

FORTRAN statements define arithmetic operations, call for input or output, and alter the sequence of program execution. Any FORTRAN statement may appear in the statement field (columns 7 to 72). Blanks (spaces) are ignored and may be used freely for appearance purposes except when they occur as alphanumeric data within a FORMAT statement, DATA statement, or literal constant. A TAB at the beginning of a line or after the statement number causes spacing to column 7. The compiler treats the first TAB after column 7 as a blank. It ignores any input line that does not contain a TAB or at least six other characters in columns 1 to 72.

Comments explaining the program may be written in any format. A line that contains the letter C in column 1 is interpreted as a line of comments. Comment lines are printed on all listings, but are otherwise ignored by the compiler. A comment line must not immediately precede a continuation line. The TAB key may be used to locate the statement field of each line.

7.4.1 Constants

The constants, variables, and expressions described below are basic to expressing data values in the FORTRAN lanugage. Five types of constants are used in OS/78 FORTRAN IV programs: integer, real, octal, logical, and Hollerith.

7.4.1.1 Integer Constants – An integer constant consists of from one to seven decimal digits written without a decimal point. Negative constants must be preceded by a minus (–) sign; however, the plus (+) sign preceding a positive constant is optional. Embedded commas and blanks are not allowed in integer constants. The following are some examples:

Examples:

```
0
+051
-440
6073
```

Integer constants must fall within the range -2^23 to 2^23-1 (-8,388,608 to 8,388,607 decimal). When used as subscripts, integer constants are taken modulo 2^12 (4096 decimal). The following are illegal as integer constants:

```
10.3     (decimal point)
5,000    (comma)
9000000  (outside acceptable range)
```

7.4.1.2 Real Constants – A real constant is an integer constant followed by a decimal point, a second string of digits, and an optional exponent. The compiler uses only the leftmost six digits, aside from leading zeros. A minus (–) sign must precede a negative constant. The plus (+) sign preceding a positive real constant is optional.

Real constants may be entered in exponential notation, as illustrated below, by specifying a positive or negative decimal value followed by the letter E and a 1 to 3 digit integer that may be positive, negative or zero. The value of the real constant is taken as the value of the decimal number preceding the letter E multiplied by that power of 10 indicated by the integer following the letter E. This notation eliminates leading and trailing zeroes from very large or very small real constants. The absolute value of any real constant must fall within the approximate range 10^-615 to 10^615 (or zero). Some examples are as follows:

```
  0.0
   .579
-10.794
  5.0E03   (i.e., 5000.)
  5.0E+3   (i.e., 5000.)
  5.0E-3   (i.e., 0.005)
  5.0E0    (i.e., 5.0)
  5.E0
```

The following are not valid real constants.

6,517.6	(comma)
131	(no decimal point or exponent)
20E1.5	(exponent must be integer)

7.4.1.3 Octal Constants – An octal constant is a string of octal digits (0 to 7 only) preceded by the letter O. Only the 12 low-order digits are used by the compiler. Octal constants are valid only in DATA statements where they are generally used to set bits for masking purposes. Some examples are as follows:

```
DATA JOB/01/32/
DATA BASE/07777/
```

7.4.1.4 Logical Constants – The two logical constants .TRUE. and .FALSE. have the internal values 1. and 0., respectively. They may be entered in DATA or input statements as .TRUE. or .FALSE. (or abbreviations .T. or .F.). The enclosing periods are part of the constant and always appear. Logical quantities are operated upon by logical operators only.

7.4.1.5 Hollerith Constants – A Hollerith constant (or literal constant) is a string of ASCII characters. It may be represented by one of two forms:

Form 1: nH character string

where n is the number of characters following the H.

Some examples are as follows:

```
SHWORDS
3H123
```

Form 2: 'character string'

Some examples are as follows:

```
'WORDS'
'123'
```

The apostrophe character that delimits a Hollerith constant in Form 2 may be included in the character string if it is immediately preceded by a second apostrophe. Thus, 'DON''T' will be stored as DON'T.

A Hollerith value may be entered in a DATA statement or FORMAT statement as a string of one to six ASCII characters per integer or real variable.

7.4.2 Variables

A variable is quantity that is represented by a symbolic name. Arithmetic statements and ASSIGN statements are used to change the value of a variable, by computation or assignment, during program execution. I/O statements

and subroutine calls can also change the value of a variable. A variable name is a string of one to six alphanumeric characters, the first of which must be alphabetic:

Valid Names	Invalid Names	
J	1ACT	(first character number)
ALPHA	STANDARD	(too long)
MAX	FILE	(space within name)
A34	#MAIN	(# not alphanumeric)

There are three types of variables: integer, real, and logical. Definitions for these types correspond to definitions of constants of the same type, that is, integer variables take on a value of from zero to any positive or negative integer in the range -8,388,607 (decimal) to 8,388,607 (decimal); real variables contain a decimal point; and so forth.

Type classification is assigned to a variable explicitly via a type declaration statement or by virtue of the initial letter of its name. A first letter of I, J, K, L, M, or N indicates an integer variable. Any other first letter indicates a real variable. The type declaration statement overrides the type indicated by the initial letter.

7.4.2.1 Arrays – Variables can be either scalar (representing a single quantity) or array (representing many quantities with one name). An entire array is identified by its name, while a single element of the array is identified by a subscript, in parentheses, following the array name.

Variable	Refers To
ARRAY (1)	The first element of a one-dimensional array named ARRAY.
B(1,3)	The element located in the first row and the third column of a two-dimensional array named B.

7.4.2.2 Subscripts – The subscripts of an array variable can be integer constants or expressions. For example, A(1), A(NEW), and A(I+1,2*K+3*J) illustrate valid subscripts. The elements of an array must be of the same type, that is, all real or all logical.

The extent of an array is determined by the dimensions it is assigned. This assignment may be made by means of a DIMENSION or COMMON statement or as part of a type declaration statement.

7.4.3 Expressions

An expression is a combination of elements (constants, subscripted or nonsubscripted variables, and function references), each of which is related to another by operators and parentheses. An expression represents one single value that is the result of calculations specified by the elements and operators that make up the expression. An expression may, itself, function as an element in another expression if it is enclosed in parentheses. The FORTRAN language provides two kinds of expressions: arithmetic and logical.

7.4.3.1 Arithmetic Expressions – An arithmetic expression is a combination of constants, variables, and function references separated by arithmetic operators and parentheses. In the absence of parentheses, algebraic operations within arithmetic expressions are performed in the following descending order:

F(x)	functions
**	exponentiation
-	unary minus
* and /	multiplication and division
+ and -	addition and subtraction
=	equals or replacement sign

Parentheses are used to change this order of precedence. An operation enclosed in parentheses is performed before its result is used in other operations. In the case of operations of equal precedence, the calculations are performed from left to right. Additional computations (such as sine, cosine, or square root extraction) may be specified via a function reference.

An arithmetic expression may consist of a single constant, variable, or function call, referred to as a basic element. For example,

```
2.71828
Z(N)
TAN (THETA)
```

Any function reference acts as a basic element in an expression, since all functions return a unique value for any given argument. The reference SQRT(4.), for example, always represents the value 2. in an expression.

Any arithmetic expression may be enclosed in parentheses and considered as a basic element. For example,

```
IFIX(X+Y)/2
(ZETA)
(COS(SIN(PI*EM)+X))
```

Compound arithmetic expressions may be formed using numeric operators to combine basic elements. For example,

```
X+3
TOTAL/A
PI*EM
```

A basic element preceded by a plus (+) or minus (-) sign is also an arithmetic expression. For example,

```
+X
-(ALPHA*BETA)
-(SQRT(-GAMMA))
```

With the exception of unary minus, no two arithmetic operators may appear in sequence. For instance, X*/Y is illegal.

Parentheses do not imply multiplication, thus (A+B) (C+D) is improper. This expression must be written as

```
(A+B)*(C+D)
```

A typical numeric expression using numeric operators and a function reference, such as the expression for one of the roots of the general quadratic equation

$$\frac{-b+\sqrt{b\uparrow 2-4ac}}{2a}$$

might be coded as

```
(-B+SQRT(B**2-4.*A*C))/(2.*A)
```

The following examples illustrate conversion of other mathematical expressions into FORTRAN expressions.

Mathematical	FORTRAN
$a+5$	A+5.0
a.b	A*B
$a^{(b+2)}$	A**(B+2)
$\left(\frac{b+d}{a}\right)^{2.5}$	((B+D)/A)**2.5

In general, only real and integer quantities may be mixed in arithmetic expressions. No other type mixing is legal. Logical variables and constants may only be operated upon by logical operators (.AND., .OR., .NOT., .XOR., .EQV.). Hollerith literals in expressions have type integer, with only the first six characters being used.

7.4.3.2 Logical Expressions – A logical expression combines logical constants, logical variables, logical function references, and logical expressions, using the logical and relational operators given in Tables 7-9 and 7-10, respectively.

Logical operators can combine only basic elements whose type is logical. Relational operators compare units of type integer or real. The value of such an expression will be of logical type (that is, .TRUE. or .FALSE.). The relational operators .EQ. and .NE. may also be used with complex expressions. Complex quantities are equal if the corresponding parts are equal.

Table 7-9 Logical Operators and Their Meanings

Logical Operator	Meaning
.NOT.expr	Has the value .TRUE. only if the expression is .FALSE., and has the value .FALSE. only if the expression is .TRUE.
expr1.AND.expr2	Has the value .TRUE. only if expr1 and expr2 are both .TRUE., and has the value .FALSE. if either expr1 or expr2 or both are .FALSE.
expr1.OR.expr2	(Inclusive OR) Has the value .TRUE. if either expr1 or expr2 or both are .TRUE., and has the value .FALSE. only if both expr1 and expr2 are .FALSE.
expr1.XOR.expr2	(Exclusive .OR.) Has the value .TRUE. if either expr1 or expr2, but not both, is .TRUE., and has the value .FALSE. otherwise.
expr1.EQV.expr2	(Equivalence) Has the value .TRUE. if expr1 and expr2 are both .TRUE. or both .FALSE., and has the value .FALSE. otherwise.

Table 7-10 Relational Operators and Their Meanings

Relational Operator	Meaning
.GT.	greater than
.GE.	greater than or equal to
.LT.	less than
.LE.	less than or equal to
.EQ.	equal to
.NE.	not equal to

The enclosing periods are part of the logical and relational operators, and must be present.

A logical expression, like an arithmetic expression, may consist of basic elements or a combination of elements, as in

```
.TRUE.
X.GE.3.14159
```

or

```
TVAL.AND.INDEX
BOOL(M).OR.K.EQ.IMIT
```

where BOOL is a logical function with one argument or a singly dimensioned logical array. A logical expression may also be enclosed in parentheses and function as a basic element. Thus, the following expressions are evaluated differently:

```
A.AND.(B.OR.C)
and
(A.AND.B).OR.C
```

No two logical operators may appear in sequence, except where .NOT.. Any logical expression may be preceded by the unary operator .NOT. as in

```
.NOT.T
.NOT.X+7.GT.Y+Z
BOOL(K).AND..NOT.(TVAL.OR.R)
```

Logical and relational operations (unless overridden by parentheses) are carried out in the following order:

```
.GT.,.GE.,.LT.,.LE.,.EQ.,.NE.
.NOT.
.AND.
.OR.
.EQV.,.XOR.
```

For example, the logical expression

```
.NOT.ZETA**2+Y*MASS.GT.K-2.OR.PARITY.AND.X.EQ.Y
```

is interpreted as

```
(.NOT.(((ZETA**2)+(Y*MASS)).GT.(K-2))).OR.(PARITY.AND.(X.EQ.Y))
```

There are 16 logical operators theoretically possible between the logical expressions. Two of them are constants (true and false) and four are unary operators (that is, the value of one of the two expressions is irrelevant to the value of the operation). These six are marked by asterisks in Table 7-11. The remaining ten operators can be most conveniently represented as shown at the right of the table, with A and B representing the two logical expressions involved.

Table 7-11 Truth Table for Logical Expressions

Expressions involved:

Operands				
A	F	F	T	T
B	F	T	F	T

Function Name	Result	FORTRAN IV Expression
* FALSE	F F F F	.FALSE.
AND	F F F T	A .AND. B
	F F T F	A .AND. .NOT. B
* A	F F T T	A
	F T F F	.NOT. A .AND. B
* B	F T F T	B
XOR	F T T F	A .XOR. B
OR	F T T T	A .OR. B
NOR	T F F F	.NOT. (A .OR. B)
EQV	T F F T	A .EQV. B
* NOT B	T F T F	.NOT. B
	T F T T	A .OR. .NOT. B
* NOT A	T T F F	.NOT. A
	T T F T	.NOT. A .OR. B
NAND	T T T F	.NOT. (A .AND. B)
* TRUE	T T T T	.TRUE.

7.4.4 Assignment Statements

A variable may be assigned a value by an assignment statement anywhere in the source program. During program execution, the most recent assignment determines the variable's value in subsequent statements. The statements that may be used to assign a value to a variable are the arithmetic and logical statements, which assign a numeric or logical value, and the ASSIGN statement, which assigns a statement number.

7.4.4.1 Arithmetic Statements – Arithmetic statements indicate computations to be performed by OS/78 FORTRAN IV.

Form v=e

where v is a variable name;

e is an expression; and

= is the replacement operator.

Effect The variable v is assigned the value of expression e.

The arithmetic statement associates a variable name with a value. This name may then be used in subsequent expressions to represent the value. Thus, if the arithmetic statement A=2 is executed first, the statement B=A+1 is equivalent to the statement B=2+1, or B=3.

Since the equal sign in the arithmetic statement does not indicate equality, but rather a replacement, statements of the form

```
I=I+1
```

are perfectly legal. The arithmetic statement is, in fact, the only means in FORTRAN by which the results of computations represented by expressions may be stored.

In the following examples, the expression to the right of the equal sign is evaluated and converted when necessary to conform to the type of the variable to the left of the equal sign. The converted value is stored in the storage location associated with the variable name to the left of the equal sign. That is, if a real expression is assigned to an integer variable, the value of the expression is converted to an integer before assignment.

Examples:

```
ANS=Y*(X**2+Z)
I=1*N
X(J)=A(J)-B(J)
P=.TRUE.
S=D.LT.5
```

The expression to be assigned must be capable of yielding a value that conforms to the type attribute of the variable to which it is being assigned. The compiler performs conversions in accordance with Table 7-12.

Table 7-12 Conversion Rules for Assignment Statements

To: From:	Real	Integer	Logical Constant	Literal Constant
Real	D	D	D	D,6
Integer	C	D	D	D,6
Logical	N	N	D	N,6

C= Conversion between integer and real.
N= Convert non-zero to 1.0 (logical truth).
D= Direct replacement.
6= Use the first character in the literal and five characters following.

7.4.4.2 The GO TO Assignment Statement – The ASSIGN statement is used in conjunction with an assigned GO TO statement to permit symbolic referencing of statements.

Form ASSIGN n to var

where n = is a statement number; and

var is an integer or real variable.

Effect The variable represents the assigned statement number and may be used in an assigned GO TO statement.

The statement number assigned must be that of an executable statement. If more than one ASSIGN statement refers to the same integer variable name, the value assigned by the last executed statement is the current value.

A variable that has obtained its value via an ASSIGN statement must be redefined via an arithmetic statement before it can be used in any context other than the GO TO statement. For example, the statement

```
ASSIGN 10 TO COUNT
```

associates the variable name COUNT with statement number 10. The following statement is then invalid:

```
COUNT=COUNT+1
```

The statement becomes valid, however, if preceded by an arithmetic assignment statement such as

```
COUNT=10
```

which assigns COUNT the integer value of 10. The use of an arithmetic assignment, however, invalidates any future use of the variable COUNT in an assigned GO TO.

Never use an assigned GO TO to transfer program control outside of the program or subprogram in which it appears.

7.4.5 Control Statements

Statements are normally executed in the sequence in which they appear in the source program. This sequence may be altered by the use of one of the following FORTRAN control statements: GO TO, IF, DO, CONTINUE, PAUSE, STOP, CALL, and RETURN. The CALL and RETURN statements, which transfer control to and from subroutines, are described in this section.

7.4.5.1 GO TO Statements – The GO TO statement transfers control directly to a specified statement. There are three forms of the GO TO statement: unconditional, computed, and assigned. A GO TO statement may appear anywhere in the executable portion of the source program except as the terminal statement in a DO loop.

Unconditional GO TO Statement

Form GO TO n

where n= the statement number of an executable statement.

Effect Control is transferred to statement n

When control is transferred by a statement of the form GO to n, the usual sequential processing continues at the statement whose number is n.

NOTE

Explicit statement numbers should not be confused with internal statement numbers (ISN's) which are sequential octal numbers assigned by the FORTRAN compiler.

Examples:

```
GO TO 50

GO TO 1020
```

Computed GO TO Statement

Form GO TO (n1,n2,...,nK),e

A comma must follow the right parenthesis.

where n1,n2,...,nK are statement numbers; and

e is a positive (nonzero) integer expression whose value is less than or equal to the number of statement numbers within the parentheses.

Effect Control is transferred to the statement whose number is eth in the list of statement numbers.

The integer expression in a computed GO TO statement acts as a switch, as in the following example:

```
GO TO (20,10,6), K
```

If K=1, control will be transferred to statement 20; if K=2, to statement 10; or if K=3, to statement 6. If K has a value less than 1 or greater than the number of statements within the parentheses, unpredictable results occur.

Assigned GO TO Statement

Form GO TO v

or

GO TO v,(n1,n2,...,nk)

where v is an integer variable; and

n1,n2,...,nk are statement numbers whose values may have been assigned to v.

Effect Control is transferred to the statement whose number is currently associated with the variable v via an ASSIGN statement.

An ASSIGN statement defines an integer or real variable as a statement number. Thus, when the statement

```
ASSIGN 10 TO LOOP
```

has been executed, control is transferred to statement 10 by the assigned GO TO statements:

```
GO TO LOOP
```

or

```
GO TO LOOP,(10,20,100)
```

either of which may be used to transfer control to whichever statement number is currently associated with LOOP.

Never use an assigned GO TO statement to transfer program control outside of the program or subprogram in which it appears.

7.4.5.2 IF Statements – An IF statement causes control to be transferred on the basis of the value of a specified expression. There are two forms of the IF statement: arithmetic and logical.

Arithmetic IF Statement

Form	IF (arithmetic expression) n1,n2,n3
where	n1,n2,n3 are statement numbers.
Effect	Control is transferred to n1 if expression <0; n2 if expression =0; and n3 if expression >0.

An IF statement transfers control to one of three statements, as shown in the model, according to the value of the expression given. For example, the following statements transfer control to statement number 30:

```
ALPHA=3
IF (ALPHA) 10,20,30
```

If fewer than three statement numbers are present, control passes to the next sequential statement for each of the missing conditions. Thus,

```
IF (ALPHA) 10
STOP
```

transfers control to 10 if ALPHA is negative; otherwise, it executes the STOP statement.

Logical expressions may be used in an arithmetic IF statement. In such statements, the logical expression is first converted to an integer.

Logical IF Statement

Form	IF (logical expression) statement
where	statement may be any executable statement except another logical IF or a DO statement.

Effect — The statement given is executed if the expression has the value .TRUE.; otherwise, the next statement in sequence is executed.

Examples:

```
LOGICAL T,F
IF (T.OR.F)X=Y+1
IF (Z.GY.X) CALL SWITCH (S,Y)
IF (K.EQ.INDEX) GO TO 15
```

7.4.5.3 DO Statement – DO statements provide for the repeated execution of a statement or series of statements.

Form — DO n i=m1,m2,m3

where — n is a statement number;

i is a nonsubscripted integer or real variable; and

m1,m2,m3 are integer or real constants or expressions.

Effect — i is set to m1 and statements following the DO statement up to and including statement n are executed repeatedly increasing i by m3 at the end of each iteration, until i is greater than m2.

The statements that are executed as a result of a DO statement are called the range. The variable i is called the index. The values m1, m2, and m3 are, respectively, the initial, limit, and increment values of the index. Note that the range of a DO need not be merely a section of straight line code following the DO statement. A control statement that causes instructions elsewhere in the program to be executed is permissible, as long as control eventually comes back to the terminal statement. When the range contains such control statements, it is called an extended range.

If m3 is omitted, an increment of 1 is assumed. A zero or negative increment is not permitted. The range of a DO is always executed at least once, regardless of the values of the limit and increment. After each execution of the range, the increment value is added to the value of the index and the result is compared with the limit value. If the value of the index is not greater than the limit value, the range is executed again using the new value of the index.

Examples:

```
DO 5 I=1,100
```
(I=100 during last iteration of DO loop)

```
DO 20 I=5,100,2
```
(I=99 during last iteration of DO loop)

```
DO 100 I=0,100,2
```
(I=100 during last iteration of DO loop)

After the last execution of the range, control passes to the statement immediately following it. This exit from the range is called the normal exit. Exit may also be accomplished by the execution of a control statement within the range.

The values of the initial, limit, and increment variables or expressions of the DO loop may be altered within the range of the DO statement. Such alteration will not affect the operation of the loop, since the values of m1, m2, and m3 are remembered by the program. Altering the index will affect the number of iterations of the loop, however. This value is available for program use as a variable. For example,

```
          DO 40 I=1,10
          TEMP=I-1
          ANUMBR=TEMP*0.1
40        ROOT=SQRT(ANUMBR)
```

In this example, the value of the index I is used as the minuend in determining the value of TEMP. Also, when a statement transfers control outside the range of a DO loop, (for example, by a GO TO or IF), the index retains its current value and is available for use as a variable. A transfer into a DO loop from outside its range may cause improper partial execution of the loop unless the transfer into the range is a return from the extended range.

The terminal statement of a DO range must not be a GO TO, DO, RETURN, STOP, PAUSE, or an arithmetic IF statement. A logical IF statement is allowed as the last statement of the range, provided that it does not contain any of the statements mentioned above. For example,

```
          DO 5 K=1,4
5         IF (X(K).GT.L)   Y(K)=X(K)
6         ...
```

In this example, the range is considered ended when, and if, control would normally pass to the statement following the entire logical IF statement. Statement 5 is executed four times whether or not the statement Y(K)=X(K) is executed. Statement 6 is not executed until statement 5 has been executed four times. Note that statement 5 would be an error if it were

```
5         IF (X(K).GT.Y(L)) GO TO 10
```

Any statement that serves as the range limit of a DO loop must not be used as the transfer point for IF or GO TO statements which are outside the DO loop. The range of a DO statement may also include other DO statements. This is referred to as nesting. The range of any nested DO statement must fall entirely within the range of the next outermost DO statement; that is, every statement in the range of an inner loop must be within the range of its enclosing outer loop. It is possible for a terminal statement to be the terminal statement for more than one DO loop, however. Figure 7-6 illustrates the order in which nested DO's are executed.

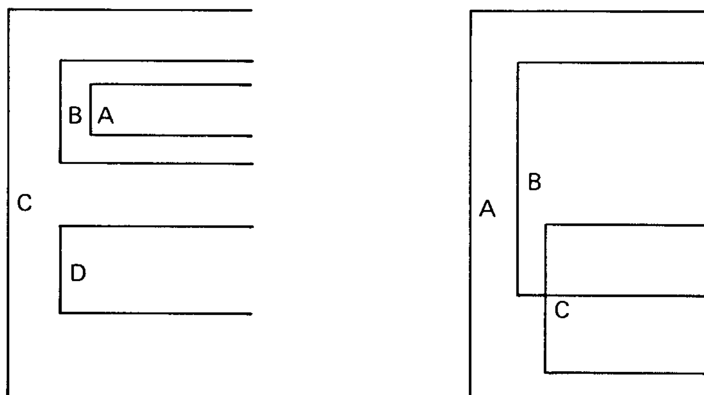


Figure 7-6 Nested DO Loops

DO loops may be nested to a depth of ten levels maximum. In calculating this depth, one implied DO in an I/O statement counts as one level whose range is the single statement, and n implied DO's within one I/O statement count as n levels, the ranges of which are all within the single statement.

7.4.5.4 CONTINUE Statement – The CONTINUE statement consists of the text

CONTINUE

and causes continuation of the normal sequence of program execution. CONTINUE is principally used as the range limit of DO loops in which the last statement would otherwise be a GO TO, IF, PAUSE, STOP, or RETURN statement. The CONTINUE statement is also used as a transfer point for IF and GO TO statements within the DO loop that are intended to begin another repetition of the loop. For example,

```
        DO 25 I=1,20
        D= D+5.0
7       IF (A-B)10,30,25
10      A=A+1.0
        B=B-2.0
25      CONTINUE
30      C=A+B
        .
        .
        .
```

A CONTINUE statement may be used as the range limit of any number of DO loops, as in this example:

```
        DO 55 I=3,5
        .
        .
        DO 55 C=1,11
        .
        .
        DO 55 V=2,6,3
        .
        .
55      CONTINUE
        .
        .
```

A CONTINUE statement serving as the range limit of a DO loop must not be used as the transfer point for IF or GO TO statements that are outside the DO loop. If it serves as the range limit of several DO loops, as above, it must not be used as the transfer point for IF or GO TO statements that are outside the innermost loop. For example,

```
        DO 20 I=1,50
        IF (K.EQ.4) GO TO 10   (incorrect)
        DO 10 I=1,50
        IF (K.EQ.4) GO TO 20   (correct)
        WRITE (5,100)I,J,K
10      CONTINUE
20      CONTINUE
100     FORMAT (3I6)
        END
```

7.4.5.5 PAUSE Statement – The PAUSE statement interrupts program execution.

Form PAUSE

or PAUSE number

where number is an integer variable or expression.

Effect The number, if any, is displayed on the terminal. Execution is suspended until a character is typed on the console.

7.4.5.6 STOP Statement – The STOP statement is placed at the logical end of a program.

Form STOP

Effect terminates the program and returns control to the Monitor.

The STOP statement terminates program execution. No continuation is possible. If no STOP statement is present in a program, a STOP occurs when control passes to the END statement in the MAIN program.

A CALL EXIT statement is equivalent to a STOP and closes tentative files at the last block written on the file. Control returns to the Monitor.

7.4.5.7 END Statement – The END statement consists of the text

END

and is placed at the physical end of a program or subprogram. In the main program, the END statement is equivalent to STOP; in a subroutine, END is equivalent to RETURN. The compiler assumes the presence of an END statement if it fails to find one before the end of the source input file. A program can not reference an END statement.

7.4.6 Data Transmission Statements

Three distinct types of data transmission statements control the transfer of data between computer memory and I/O devices: data description (FORMAT) statement, input/output (READ and WRITE) statements, and device control (BACKSPACE, REWIND, and END FILE) statements.

7.4.6.1 FORMAT Statement – The FORMAT statement describes the form and arrangement of data on a record.

Form FORMAT (spec1,spec2,.../...)

where spec1,spec2 define consecutive series of characters within a record;

/ is the end of the record description; and

) is the end of a statement.

Effect Specifies either the type of conversion to be performed between the internal and external representation of data or the format of fixed data.

A FORMAT statement must have an explicit statement number that is used in READ or WRITE statements for reference.

The field specification (spec) is one of the following:

nAw
nEw.d
nFw.d
nGw.d
nH
nIw
nLw
-sP
Tw
nX
'string'
"string"

where

n is an unsigned nonzero integer stating the number of times the field specification is to be repeated;

s is scale factor;

A,E,F,G,H,I,L,P,T,X are types of conversion;

w is nonzero, unsigned integer constant specifying width of field; field width must be large enough to provide for all characters (including decimal point, sign, and exponent) necessary to constitute the data value plus any blank characters needed to separate it from other data values; the data value within a field is right-justified; if the value is too large for the field, the field is filled with asterisks; and

.d is unsigned integer constant (may be zero) specifying number of digits to the right of the decimal point or, for G conversion, the number of significant digits.

Field specifications must be wirtten in the same sequence as the data record being described, except when the T specification is used.

A FORMAT statement may describe one or more records, each of which can consist of one or more field specifications. The slash character (/) indicates that a new record is being described. For example, the statement

```
FORMAT (G10.2/I5,2F8.4)
```

is equivalent to

```
FORMAT (G10.2)
```

for the first record, and

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

for the second record. Field specifications are separated by commas as shown above. The separating comma may be omitted when a slash is used. When n slashes appear at the end or beginning of a format, n blank records may be written on output or n records skipped on input. When n slashes appear in the middle of a format, n-1 blank records are written or n-1 records skipped.

For example,

```
FORMAT (I6,///,2F5.1)
```

where / / / indicates that two records are to be skipped.

A group of field specifications may be repeated by enclosing the group in parentheses and preceding the enclosed group with a repetition number as shown in the following example:

```
FORMAT (3(I5,F10.3))
```

Both the slash and the closing parenthesis at the end of the format indicate the termination of a record. If the list of an input/output statement dictates that t ansmission of data is to continue after the closing parentheses of the format is reached, the format is repeated starting with that group repeat specification terminated by the last preceding right parenthesis, or, if none exists, then to the first left parenthesis of the format specification. Thus, the statement:

```
FORMAT (F7.2,3(I2,2(I3,E9.3)I7))
               ↑               ↑↑
               |               | Terminator
               Group Repeat    Last preceding
               specification   right parenthesis
```

causes

(F7.2,3(I2,2(I3,E9.3)I7)

to be used on the first record, and the format

3(I2,2(I3,E(.3)I7)

to be used on succeeding records.

As a further example, consider the statement

```
FORMAT (F7.2(2(E15.5,E15.4),I7))
```

The first record has the format

(F7.2)

and successive records have the format

(2(E15.5,E15.4),I7)

FORMAT statements may be placed anywhere within the executable portion of the source program. Unless the FORMAT statement contains only Hollerith data for direct input/output transmission, it will be used in conjunction with the list of a data transmission statement. Because FORMAT statements are referenced by READ or WRITE statements, each FORMAT statement must have an explicit statement number.

The ASCII character string comprising a format specification may be stored as an array. Input/output statements may then refer to the format by giving the array name, rather than the statement number of a FORMAT statement.

The stored format has the same form as a FORMAT statement excluding the word FORMAT. The enclosing parentheses are required.

Field specifications in a FORMAT statement should be of the same type as the corresponding items in the I/O list; that is, integer quantities require integer (I) conversion, etc. There are three types of field specifications: numeric, logical, and alphanumeric (including Hollerith). In addition, a blank field description may be given to skip portions of an input record or to embed blanks within an output record.

Numeric Fields – Numeric fields are specified by one-letter codes (E, F, G, or I) that designate the type of conversion to be performed. Two parameters may appear in a numeric field description, depending on the field type; (1) an integer (w) specifying the field width (which may be greater than required to provide for blank columns between numbers); and (2) an integer (d) specifying the number of decimal places to the right of the decimal point, or, for G conversion, the number of significant digits. Decimal points are not permitted in I conversion. (For E, F, and G input, the position of the decimal point, if present in the external field, takes precedence over the value of d in the format.) Conversion codes and the corresponding internal and external forms of the numbers are listed in Table 7-13. Numeric fields are right-justified with the addition of leading spaces and, if necessary, trailing zeroes.

Single precision I/O specifications will transfer a maximum of six decimal digits of accurate data.

Table 7-13 Numeric Field Codes

Conversion Code	Internal Form	External Input Form	External Output Form
E	Real	Decimal number with or without a decimal point or exponent field.	Decimal number and an E exponent with a decimal point field.
F	Real	Decimal number with or without a decimal point or exponent field.	Decimal number with a decimal point.
G	Real	Decimal number with or without a decimal point or exponent field.	Decimal number with a decimal point and with or without an E exponent field.
I	Integer	Decimal number without a decimal point or exponent.	Decimal number without a decimal point or exponent.

Four numeric field specification forms are allowed:

1. Ew.d,
2. Fw.d,
3. Iw, and
4. Gw.d.

For example,

```
FORMAT (I5,F10.2)
```

could be used to output the following line on the output listing

bb32bbbb-17.60

where b represents a blank. Since there is no carriage control character in the statement, the first character is interpreted as a carriage control character and is not printed.

The G format is the general format code used to transmit data having a specific number of significant figures, no matter what the magnitude of the number. This format is intended to allow use of the simplest output format that can express the desired value in the space allowed. The rules for input are the same as those for E format.

The form of the output conversion (E or F) is a function of the magnitude of the data being converted. Table 7-14 shows the magnitude of the internal data, M, and the resulting method of conversion.

Table 7-14 Conversion Under G Format

Magnitude of Data	Resulting Conversion
0.1<=M<1	F(w-4).d,4X
1<=M=<10	F(w-4).(d-1),4X
.	.
.	.
.	.
10(d-2)<=M<10(d-1)	F(w-4).1,4X
10(d-1)<=M<10(d)	F(w-4).0,4X
All others	Ew.d

Scale factors may be specified for E, F, and G conversion. A scale factor is of the form

sP

where s is a signed or unsigned integer that specifies the scale factor; and P is the identifying character.

For F type conversions, the scale factor specifies a power of ten, so that

external number = (internal number)*$10^{(\text{scale factor})}$

For E conversions, the scale factor multiplies the fraction by a power of ten, but the exponent is decreased accordingly leaving the number unchanged except in form. For example, if the statement

```
FORMAT (F8.3,E16.5)
```

corresponds to the line

bb26,451bbbb-0.41321E-01

then the statement

```
FORMAT (-1PF8.3,2PE16.5)
```

would correspond to the line

bbb2.645bbb-41.3215E-03

For G type output conversions, the scale factor is not used unless the magnitude of the number is such that E format is used. In input operations, the scale factor is not used if there is an exponent in the external field.

When no scale factor is specified, a scale factor of zero is assumed. Once a scale factor has been specified, however, it holds for all subsequent E, F, and G type conversions within the same format unless another scale factor is encountered. A zero scale factor may be resumed via an explicit specification. The scale factor is initially 0. Scale factors have no effect on I type conversions.

Logical Fields – Logical data can be described in a manner similar to numeric data. A logical field description has the form:

Lw

where L is the conversion code character and w is an integer specifying the field width. The data is transmitted as the value of a logical variable in the input/output list. On input, the first non-blank character in the data field must be .T., T, .F. or F; the value of the logical variable will be stored as .TRUE., .TRUE., .FALSE., or .FALSE., respectively. Leading blanks and the period preceding the T or F are ignored. If the data field is blank, a value of false will be stored. On output, w-1 blanks followed by the letter T or F, according to the variable's value, will be transmitted. For example, if the specification were L10, the output for the value .TRUE. would be

bbbbbbbbbT

Hollerith Data A Conversion – Hollerith data that is to be stored by the program is specified by

Aw

where A is the conversion code character and w is the number of characters in the field. The alphanumeric characters are transmitted as the value of a variable in an input/output list. The variable may be of any type. The following sequence causes four characters to be read and placed in memory as the value of the variable V:

```
      READ (2,5)V
5     FORMAT (A4)
```

The value of w is limited to the maximum number of characters that can be stored in the space allotted for a single variable. If w exceeds this amount, the extraneous rightmost characters are lost on input, and on output the characters are lost on input, and on output the characters after the sixth are not significant. If w is less than the number of characters that can be stored in the space allotted to the variable, on input the characters are left-justified and blank-filled on the right of each list item. On output the leftmost w characters in the variable are transmitted to the output field.

Hollerith Data H Conversion – Hollerith data that is not changed by the program is specified by one of two forms. One, called H conversion, is

nH

where H is the control character; and n is the number of characters in the string (including blanks). For example, the format in the statement below can be used to print PROGRAM COMPLETE on the output listing.

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

Referring to this format in a READ statement causes the 17 characters to be replaced with a new string of characters from the input file.

In the second form, the Hollerith data is simply enclosed in single quotes. The output result is the same as in H conversion; on output the characters between the quotes (including blanks) are written as part of the output data, as with Hollerith constants. For example,

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

A single quote within the data is represented by two successive quotes.

A Hollerith format field may be placed among the other fields of the format. For example, the statement:

```
FORMAT (I5,7H FORCE=F10.5)
```

can be used to output the line

bbb22bFORCE=bb17.68901

Note that the separating comma may be omitted after a Hollerith format field. The legal characters in a Hollerith field are the 64-character 6-bit subset of ASCII, with the exception of @, which must not be used.

Carriage Control – The first character of each ASCII record controls the spacing of the line printer or terminal. This character may be established by beginning a FORMAT statement for an ASCII record with 1Hc, where c is the desired control character. The line spacing actions, listed below, occur before any printing:

Character	Effect
blank	Advance carriage to next line.
0 (zero)	Skip a line (double space).
1 (one)	Form feed – go to top of next page.
+ plus	Suppress skipping – overprint previous line (used only for hard copy output).

For example, the following program moves the line printer paper to the top of the next sheet (1H1) and prints b3.5 on the first line.:

```
        A=14.
        X=(A-5.25)/2.5
        WRITE (4,100)X
100     FORMAT (1H1,F4.1)
        STOP
        END
```

If any unexpected character appears first in the FORMAT statement, it is processed as a blank.

It is often desirable to print a prompting sequence, such as a question, to which a response is to be entered on the same line. To cause such prompting, a dollar sign ($) is included at the end of a FORMAT statement that is

associated with a WRITE statement that has been satisfied. This will inhibit the carriage return and line feed before the next input. For example,

```
          A=5
          WRITE(4,100)A
          READ(4,200)B
100       FORMAT(' SAMPLE NO.',I2,' IS:',$)
200       FORMAT(A6)
          WRITE(4,200)B
          END
```

The output is

```
SAMPLE NO.  5 IS: RED
RED
```

Record Layout Specification – Input and output can be made to begin at any position within a FORTRAN record by use of a field description of the form

Tw

where T is the spacing control character; and w is an unsigned integer constant specifying the character position in a FORTRAN record where the transfer of data is to begin.

For printed output, w corresponds to the (w-1)th print position, since the first character of the output buffer is a carriage control character and is not printed. (A blank carriage control indicator is assumed.) For example,

```
FORMAT (T30,'BLACK'T50,'WHITE')
```

causes BLACK to appear in columns 29-33 and WHITE to appear in columns 49-53. The statement

```
FORMAT (T50,'BLACK',T30'WHITE')
```

causes BLACK to appear in columns 49-53 of one line and WHITE to appear in columns 29-33 of the following line. The two lines will constitute two separate records if later read as input. On input, the statement:

```
1         FORMAT (T35,F5.0)
          READ (3,1)X
```

causes the first 34 characters of the input data to be skipped, and the next five characters to be used as the value of X. If an input record containing

ABCbbbXYZ

is read with the format specification

```
FORMAT (T7,A3,T1,A3)
```

then the characters XYZ and ABC are read, in that order. The T-code can be used in a FORMAT statement with any other type of format code.

Blanks can be introduced into an output record or characters skipped on an input record by use of the specification:

nX

where the spacing control character is X and n is the number of blanks or characters skipped and must be greater than zero. For example, the statement

```
FORMAT (5H STEPI5,10X2HY=F7.3)
```

can be used to output the line

STEPbbb28bbbbbbbbbbY=b-3.872

7.4.6.2 DEFINE FILE Statement – The FORTRAN program may read or write chosen records from a direct-access file without reading intermediate records. In the more common sequential access, the FORTRAN program must read or write each record in turn until the correct record is found. No DEFINE FILE statement is required for sequential access to mass storage files.

The DEFINE FILE statement is required so that mass storage files may be referenced as direct access files by input/output statements:

Form DEFINE FILE a1(b1,c1,U,v1),a2(b2,c2,U,v2), . .

where a is an integer constant or integer variable name that is the symbolic designation for this file specified to the run-time system;

b is an integer expression that defines the number of records in the file;

c is an integer expression that defines the length of each file record; each integer or real variable or constant is stored in floating point notation that requires three storage locations per value;

U is a fixed argument designating that the file is unformatted; and

v is an integer variable name, called the associated variable, which is set at the conclusion of and input/output operation on the file to point to the next record.

Effect Describes a mass storage file for use with input/output statements.

The associated variable (v) in a DEFINE FILE statement is used to maintain an index of records processed. It is set automatically after an input/output statement is executed. The statement

```
DEFINE FILE 1(1000,100,U,IV1)
```

specifies a 1000-record file, each record of which is 100 floating point variables long. The variable IV1 will maintain an index of records processed, providing a pointer to the next record.

The symbolic file designation (a) cannot be passed as a dummy argument to a DEFINE FILE statement in a subroutine.

7.4.6.3 Input/Output Statements – The input/output statements, READ and WRITE, govern transfer of data records between internal storage and peripheral devices. Each statement can contain an input/output list naming the variables and array elements to be given values on input or whose values are to be transmitted on output. Both formatted and unformatted records can be transmitted. A formatted record requires the use of a format specification.

Input/Output Lists – An input/output list contains variable names and array elements whose values will be assigned on input or written on output. Constants are not allowed as list items. During input, the new values listed can be used in subscript or control expressions for variables appearing later in the list. For example,

```
READ (6,100)L,A(L),B(L+1)
```

reads a new value for L and uses this value in subscripts of A and B.

The transmission of array variables may be controlled by indexing similar to that used in the DO statement. This is called an implied DO and includes as a list element a parenthesized list of control variables followed by the index control. For example,

```
READ (7,10) (X(K),K=1,4)
```

is equivalent to:

```
READ (7,50)X(1),X(2),X(3),X(4)
```

The indexing may be compounded by nesting the implied DO's as in the following:

```
READ (9,70)   ((MASS(K,L),K=1,4),L=1,5)
```

The above statement reads in the elements of array MASS in the following order:

MASS(1,1),MASS(2,1), . . . , MASS(4,1),MASS(1,2), . . . ,MASS(4,5)

If an entire array is to be transmitted, the indexing may be omitted and only the array name written. The array is transmitted in order of increasing subscripts with the first subscript varying most rapidly. Thus, the example above could have been written

```
READ (7,75) MASS
```

assuming that the array MASS is dimensioned MASS(4,5). The same statement can transmit integer and real quantities. If the data to be transmitted exceeds the items in the list, the extra data is ignored.

Input/Output Records – All data is transmitted by an input/output statement in terms of records. The maximum amount of information in one record and the manner of separation between records depends on the medium. On a terminal, a record is one line; for ASCII records, the amount of information is specified by the FORMAT reference and the I/O list.

Each execution of an input or output statement initiates the transmission of a new data record. If an input/output statement requests less than a full record of information, the unrequested part of the record is lost and cannot be recovered by another input/output statement without repositioning the record. Repositioning is not possible on all devices, however. If an input/output statement requires more than one ASCII record of information, successive records are read in sequence.

READ Statement

Form	READ (u,f) list	formatted READ
	READ (u,f)	
	READ (u) list	unformatted READ
	READ (u)	
	READ (a'r) list	direct access mass storage READ

where u is an input unit designation;

f is a format statement reference number;

list is an I/O list of variable names;

a is a symbolic mass storage file number (unsigned integer constant or integer variable);

' designates direct access; and

r is the record number in which transfer begins (integer expression).

Effect Input is performed according to the arguments of the READ statement.

The unit designation (u) referred to in READ statements must be an integer in the range 1 to 9. If device specifications are not given, the system assumes the standard device designations mentioned in Section 7.2.3 describing the run-time system. Thus, READ (4,11) could refer to input from the terminal. Formatted and unformatted records should not be mixed on a single unit.

A formatted READ statement causes information to be read from the specified unit and put in memory. The data are converted from external to internal form as sepcified by the referenced FORMAT statement. If an I/O list is provided, the data are stored as the values of the listed variables. The second form of the formatted READ statement is used if the data are transmitted directly into the specified format. For example,

```
READ (5,50)A,B,C
READ (4,100)
```

Detection of end-of-file during input causes a fatal run-time system error unless the library subroutine CHKEOF is called. CHKEOF should only be used with formatted I/O involving a single record.

A comma encountered on input signals the end of the current field and causes the next input character to be read as the first character of the next input field.

An unformatted READ statement causes binary information to be read from the unit designated and stored in memory as values of the variables in the I/O list, if any. If the record contains more words than the list requires, that part of the record is lost. If more items are in the list than are in one record, additional records are read until the list is completed.

A direct access READ statement provides random access to fixed-length records in a mass storage file. The file whose records are to be read must be defined by a DEFINE FILE statement. For example,

```
DEFINE FILE ATC(100,100,U,FT)
READ (ATC'5)ARRAY
```

WRITE Statement

Form		
	WRITE (u,f) list WRITE (u.f) WRITE (u) list	formatted WRITE
	WRITE (u)	unformatted WRITE
	WRITE (a'r) list	Direct access mass storage WRITE

where u is an output unit designation (unsigned integer constant or variable);

f is a format statement number;

list is an I/O list of variable names;

a is a symbolic mass storage file number;

' designates direct access; and

r is an associated variable (record pointer).

Effect Output is performed as specified by the arguments of the WRITE statement.

The unit designation (u) referred to in WRITE statements may be an integer in the range 1 to 9. Unit designations are assigned via a device specification command to the run-time system.

A formatted WRITE statement may appear with or without an I/O list. If a list is provided, the values of the variables in the list are read from memory and written on the unit designated in ASCII format. The data is converted to external form as specified by the designated FORMAT statement. If no list is supplied, information is read directly from the specified format statement and written on the designated unit in ASCII format. For example,

```
          WRITE (4,100)
100       FORMAT ("    OUTPUT RECORD")
```

will produce the following record on unit 4:

```
    OUTPUT RECORD
```

For an unformatted or direct-access WRITE, the values of the variables in the list are read from memory and written on the unit designated in binary format. A record holds 85 real or integer variables. If the list elements fill more than one record, successive records are written until the list is completed. If the list elements do not fill the record, the remaining part of the record contains undefined data. Thus, two records are used if the list contains 100 variables: one record contains 85 variables and the second record contains 15 variables and undefined data. For example,

```
    DIMENSION X(200)
    WRITE (6) X
```

will produce three records on unit 6, the first will contain X(1) to X(85); the second will contain X(86) to X(170), and the third will contain X(171) to X(200).

A direct access WRITE statement outputs a fixed-length record directly into a mass storage file. The file must have been defined previously via the execution of an appropriate DEFINE FILE statement. This means minimum record size is one block. For example,

```
             DIMENSION RAY(10)
             DEFINE FILE 1(12,10,U,J)
             J=1
             DO 5 I=1,12
             READ (4,100) RAY
100          FORMAT (10F8.0)
             WRITE (1'J) RAY
  5          CONTINUE
             STOP
             END
```

7.4.6.4 Device Control Statements – There are three device control statements: BACKSPACE, END FILE, and REWIND, that apply to diskettes. Their forms and effects are listed in Table 7-15.

Table 7-15 Device Control Statements

Statement	Effect
BACKSPACE u	If the next record, which would be read or written on unit (u), is n, BACKSPACE repositions the unit so that the next record to be read or written will be n-1. If the first record is the next record to be read or written, the BACKSPACE statement has no effect.
REWIND u	Repositions the designated unit (u) to the beginning of the file. If the unit is at the beginning of the file, the REWIND statement has no effect.
END FILE u	Writes an END-OF-FILE character in the specified file (u), provided that the file has been written on by a formatted WRITE. END FILE does not execute a REWIND.

7.4.7 Specification Statements

Specification statements may be divided into three categories:

1. Storage specification statements (DIMENSION, COMMON, and EQUIVALENCE) that give the compiler storage allocation instructions;
2. Data specification statements (DATA and BLOCK DATA) that are used to enter values; and
3. Type declaration statements (INTEGER, REAL, and LOGICAL) that specify the type of variable.

7.4.7.1 Storage Specification Statements Dimension Statement –

Form DIMENSION name1 (u1,...,u7), name2 (v1,...,v7),...

where u1,...,u7 and v1,...,v7 are the maximum values of the subscripts they represent, up to a maximum of seven subscripts.

Effect The array name assigns the type to the array. Storage is allocated according to the dimensions given.

Each array specification gives the array name and the maximum size that each of its subscripts may assume. Array size is limited to 4095 elements. Each size specification must be a nonzero positive integer constant. For example,

```
DIMENSION A(10),B(4,6),X(5,5,5)
```

defines A as a one-dimensional array variable with storage locations for 10 words. Array B is defined as a two dimensional array with storage for 24 words (4x6). Array X is a three-dimensional array with 125 words (5x5x5).

In certain cases involving subroutines, a dimension may be an unsubscripted integer parameter.

Any number of arrays may be declared in a single DIMENSION statement. Each array variable appearing in the program must represent an element of an array declared in a DIMENSION statement, unless the dimension information is given in a COMMON or type statement.

Dimension information may appear only once for a given variable. The DIMENSION statement must precede any reference to the variable including reference in a DATA or EQUIVALENCE statement. Type declarations or COMMON statements may appear anywhere in a program unless they include dimension information.

A subprogram can establish adjustable arrays at execution time if both the array name and the subscript size are expressed as dummy arguments in the subroutine, as in

```
SUBROUTINE WHAT(A,X,Y,Z)
DIMENSION A(X,Y,Z)
```

To do this, the programmer must establish A, X, Y and Z as required arguments. The dummy array must not exceed the dimensions of the main program array but may be smaller if the call provides lower subscript sizes than those of the main program dimensioning or if the initial array element referenced is not the beginning of the main program array.

COMMON Statements

Form	COMMON/block1/a,b,c/block2/d,e,f/...
where	block1, block2,..., are the block names; and a,b,c,d,e,f are the variables to be assigned to each block.
Effect	Specified variables or arrays are stored in an area available to other programs.

By means of COMMON statements, the data of a main program and/or the data of its subprograms can share a common storage area. The common area can be divided into separate blocks identified by block names that may not be the same as any program variable names. A block is specified as follows:

/block name/var1, var2,...

The variables following the block name indicate scalar or array variables assigned to the block, They are placed in the block in the order in which they appear in the block specification. For example, the statement:

```
COMMON/R/X,Y,T/C/U,V,W,Z
```

indicates that the elements X, Y, and T are to be placed in block R in that order, and that U, V, W, and Z are to be placed in block C. Variables whose names appear in the formal parameter list must not also appear in COMMON declarations within the subroutine. A COMMON block may not have the same name as a variable in the same program. In addition, a COMMON block may not have the same name as any subprogram used at the same time as the COMMON block.

Block entries are linked sequentially throughout the program, beginning with the first COMMON statement. For example, the statements

```
COMMON/D/ALPHA/R/A,B/C/S
COMMON/C/X,Y/R/U,V,W
```

have the same effect as the statement

```
COMMON/D/ALPHA/R/A,B,U,V,W/C/S,X,Y
```

One block of COMMON storage, referred to as blank COMMON, can be left unlabeled. Blank COMMON is indicated by two consecutive slashes. For example,

```
COMMON/R/X,Y//B,C,D
```

indicates that B, C, and D are placed in blank COMMON. The slashes may be omitted when blank COMMON is the first block of the statement, as in

```
COMMON B,C,D
```

Storage allocation for blocks of the same name begins at the same location for all programs executed together. For example, if a program contains

```
COMMON A,B/R/X,Y,Z
X=3
```

as its first COMMON statement, and a subprogram has

```
COMMON/R/U,V,W//D,E,F
```

as its first COMMON statement, the quantities represented by X and U are stored in the same location, that is, X and U both refer to the third variable. A similar correspondence holds for A and D in blank COMMON.

COMMON blocks may be of any length, subject to the limitations on available memory.

Array names appearing in COMMON statements may have dimension information appended if the arrays have not been declared via a DIMENSION statement or a type declaration. For example:

```
COMMON ALPHA,T(15,10,5),GAMMA
```

specifies the dimensions of the array T while entering T in blank COMMON. If array dimensions are not defined in a COMMON statement, they must be defined in some other type statement.

EQUIVALENCE Statement

Form EQUIVALENCE (v1,v2,...,), (vk, vk+1,...),...

where v1,v2,..., vk are the variable names.

Effect The variables within the parentheses identify the same storage location.

The following example

```
EQUIVALENCE(RED,BLUE)
```

specifies that the values of the variables RED and BLUE are stored in the same location. If used at different times, multiple variables with different values can occupy the same storage location or if used at the same time, multiple variables can be assigned the same value through the use of EQUIVALENCE.

The master variable in an equivalence group is either the variable in the group that is in COMMON (only one such variable per group is legal) or the first variable in the group. All the other variables of an equivalence group are considered subordinate and can only appear in one group. The master of an equivalence group should be large enough to encompass all the subordinate variables equivalenced to it.

The subscripts of array variables in an EQUIVALENCE statement must be integer constants as shown in the following example:

```
EQUIVALENCE(X,A(3),Y(2,1,4)),(BETA(2,2),ALPHA)
```

The formal parameters of a subroutine must not appear in EQUIVALENCE statements within that subprogram.

The variables assigned by an EQUIVALENCE statement must be within the same main program or within the same subprogram.

Variables EQUIVALENCE and COMMON Statements – Variables may appear in both COMMON and EQUIVALENCE statements, but no two quantities in COMMON may be set equivalent to one another.

Quantities placed in a COMMON block by means of EQUIVALENCE statements can cause the end of the COMMON block to be extended. For example, the statements

```
COMMON/R/X,Y,Z
DIMENSION A(4)
EQUIVALENCE(A,Y)
```

cause the COMMON block R to extend from X to A(4), arranged as follows:

```
X
Y A(1)
Z A(2)
  A(3)
  A(4)
```

EQUIVALENCE statements that would require extension of the start of a COMMON block are not allowed. For example, the sequence

```
COMMON/R/X,Y,Z
DIMENSION A(4)
EQUIVALENCE(X,A(3))
```

is not permitted, since it would require A(1) and A(2) to extend the starting location of block R.

Care must be exercised when using EQUIVALENCE and COMMON statements.

Data Statement

Form DATA var list1/val list1/,var list2/val list2/,...

where var list contains a string of variables separated by commas; and

/val list/ contains a string of data items separated by commas.

Effect A value from val list is assigned to the corresponding variable in var list.

The DATA statement is used to supply initial or constant values for variables. The specified values are compiled into the object program, and become the values assumed by the variables when program execution begins. Such values may also be provided via a BLOCK DATA subprogram. It is recommended that variables in COMMON be initialized only by means of a BLOCK DATA subprogram.

Elements in the variable list may be either single subscripted or unsubscripted variables, or the name of an entire array.

When an entire array is given, data values must be specified for every element of the array. Data elements are stored in the array in the same order as that used for the data transmission and storage arrays, that is, in order of increasing subscripts with the first subscript varying most rapidly.

Allocation to memory locations in the array stops when

1. the data item list is exhausted, or
2. data items have been allocated to the entire array; if so, additional data items will be allocated to succeeding variables listed.

When Hollerith or literal constants are encountered in the values list, they are assigned to the associated variables in the same manner as in assignment statements.

A Hollerith value in a data statement will occupy (n+5)/6 words of storage, with any partial words filled to the right with blanks.

The data items following each list of variables must have a one-to-one correspondence with the variables of the list, since each item of the data specifies the value given to its corresponding variable.

Data items assigned may be numeric, Hollerith, octal, or logical constants. For example,

```
DATA ALPHA, BETA/5,16,E-2/
```

specifies the value 5 for ALPHA and the value 0.16 for BETA. Any item of data may be preceded by an unsigned nonzero integer constant followed by an asterisk. This notation indicates that the item is to be repeated. For example,

```
DATA A(1),A(2),A(3)/3*0./
```

specifies the value zero for array elements A(1)-A(3). As another example,

```
DIMENSION A(2,2),B(3)
DATA A,B/2*1.0,3*2.0,3.0,4,/
```

will initialize

A(1,1) and A(2,1) to 1
A(1,2), A(2,2) and B(1) to 2
B(2) to 3, and B(3) to 4

7.4.7.2 Type Declaration Statements

Form type v1, v2, v3,...

where type may be INTEGER, REAL, LOGICAL; and

v1, v2, v3,... represent variables.

Effect All variables in the list are assigned the given type.

A variable can appear in only one type declaration statement. Type declaration statements override any implicit type specifications determined by the first letter of a variable. Type statements can be used to give dimension specifications for arrays. Adjustable arrays in subprograms can also be defined via type declaration statements. Each variable or function name in a type declaration statement is defined to be of that specific type throughout the program; the type cannot change.

Examples:

```
INTEGER ABC,IJK,XYZ
REAL A(2,4),I,J,K
```

7.4.8 Subprogram Statements

The use of subprograms allow a statement or group of statements to be written once and then referenced whenever the implemented operation is to be executed. The use of subprograms saves programming time and computer memory. There are three categories of subprograms in FORTRAN: FUNCTION subprograms, SUBROUTINE subprograms, and BLOCK DATA subprograms.

Functions and subroutines consist of one or more FORTRAN statements, which may be invoked by name, and, as appropriate, with values upon which they are to operate. A function differs from a subroutine in that it is always called with at least one parameter, and always returns at least one value as the value to be used for the function reference in the expression in which it occurs. A subroutine may be called with or without parameters and may return values only through its parameters or via COMMON. BLOCK DATA subprograms contain specification statements only and are used to specify initial values for variables in COMMON.

The transmission of arguments between a subprogram reference and the subprogram itself is accomplished by the use of dummy variables within the subprogram definition. The dummy variables in the subprogram are listed in the subprogram definition statement. References to the subprogram may then supply values for these arguments in the same order and be substituted for them whenever they appear in the subprogram.

7.4.8.1 Functions – An internal function is defined via a form of the arithmetic assignment statement and may be referenced only by the program in which it is defined.

Form name (arg1,...) = expression

where name is the function name;

arg1,... are dummy arguments; these variables will be altered whenever the function is used and should not be referenced elsewhere in the program; and

expression is the function definition.

Effect Defines an internal function.

An arithmetic statement function definition is a single statement. The expression which defines the function may include dummy arguments, ordinary variables, external functions and previously defined internal functions.

In the following definition,

ACOSH(X) = (EXP(X/A) + EXP(- X/A))/2.

X is a format argument and A an ordinary variable. When the function is referenced, the current value of A and the supplied value of X will be used to evaluate it. All function definitions of this type must precede the first executable statement of the program in which they appear, and follow the last specification statement appearing in the program.

7.4.8.2 FUNCTION Statements – An external function, one which may be referenced by other programs, is defined via the FUNCTION statement. A function reference may only appear within an expression and must, like other elements of expressions, have a specified type. Type may be specified in the definition itself or via any other FORTRAN type specification facility.

Form	t FUNCTION name (arg1,...)
where	t is an optional type specification (for example, REAL)
	name is the function name; and
	arg1,... are dummy arguments.
Effect	Defines an external function subprogram.

The function name must be a legal symbol that is assigned a value within the subprogram definition. The last value assigned to this name is the function's value. There must be at least one argument and arguments must agree in number, order, and type with actual arguments given by the calling program. A maximum of six dummy arguments may be given in a FUNCTION statement. For example,

```
        FUNCTION M(X)
        IF (X.GE.12.0) GO TO 50
        M=10.0 * 2.5
        RETURN
50      M=.5 * 5.0
        RETURN
        END
```

Dummy arguments may represent the following elements in the function definition: expressions, alphanumeric strings, array names, or elements and subprogram names. Dummy arguments that represent array names must appear within the subprogram either in a DIMENSION statement or in one of the type statements that provide dimension information. Dimensions given as constants must not exceed the dimensions of the corresponding arrays in the calling program. Dimensions given as dummy variables may be used to specify adjustable dimensions for array name arguments. For example, in the statement sequence

```
FUNCTION TABLE (A,M,N,B,X,Y)
.
.
.
DIMENSION A(M,N),B(10),C(50)
```

the dimensions of array A are specified by the dummy arguments M and N, while the dimension of array B is given as a constant. The various values given for M and N by calling the program must be within the limits of the actual arrays that the dummy array A represents. Various arrays may be substituted for A. These arrays may each be of different size. Dummy dimensions may only be given for dummy arrays. Note in the example above that array C, which is not a dummy argument, must be given absolute dimensions. A dummy argument can not appear in an EQUIVALENCE statement in the function subprogram.

A function may modify any arguments that appear in the FORTRAN arithmetic expression calling the function. The only FORTRAN statements not allowed in a function subprogram are SUBROUTINE, BLOCK DATA, and another FUNCTION statement.

A function is called by placing the function name in the program. Control then passes to the function subroutine. The quantity resulting from the function subroutine replaces the function name in the calling expression.

7.4.8.3 Subroutine Subprograms – A subroutine subprogram is defined external to the program that references it. Subroutine definition is initiated by a SUBROUTINE statement. A subroutine is invoked by a CALL statement and returns control to the calling program by means of a RETURN statement.

SUBROUTINE Statement

From	SUBROUTINE name or SUBROUTINE name (arg1,...)
where	name is subroutine name; and arg1,... are optional dummy arguments.
Effect	The program that follows is declared a subroutine program.

The arguments in the parenthesized list are dummy arguments representing the arguments of the subprogram. The dummy arguments must agree in number, order, and type with the actual arguments used by the calling program. A subroutine subprogram need not have any arguments at all; a maximum of six dummy array arguments are allowed. When supplied, they may be expressions, alphanumeric strings, array names, array elements, scalar variables, and subprogram names.

Dummy variables representing array names must be dimensioned within the subprogram by a DIMENSION or type declaration statement. As in a function subprogram, either constants or dummy identifiers may be used to specify dimensions in a DIMENSION statement. The dummy arguments may not appear in an EQUIVALENCE or COMMON statement in the subroutine program.

A subroutine or function subprogram may use one or more of its dummy arguments to represent results. For example,

```
SUBROUTINE COMPUT (A,B,ANS)
```

might require the user to supply numeric values for A and B to be computed, and a variable for ANS in which to store the results. The only FORTRAN statements not allowed in a subroutine subprogram are FUNCTION, BLOCK DATA, and another SUBROUTINE statement.

Constants in call lists of subroutines and function subprograms are not protected. Therefore, a function such as the following will result in erroneous values:

```
FUNCTION FNC1 (Y)
FNC1=100.0
Y=Y+2.0
RETURN
END
```

CALL Statement

Form CALL name or
CALL name (argl, . . .)

where name identifies a subprogram; and

argl, . . . are actual arguments.

Effect Control is transferred to the subroutine subprogram.

The arguments of a CALL statement can be expressions, array names, array elements, scalar variables, alphanumeric strings, or subprogram names; arguments may be of any type, but must agree in number, order, type, and array size (except for adjustable arrays, as discussed under the DIMENSION statement) with the corresponding arguments in the SUBROUTINE statement of the called subroutine. A subroutine cannot be referred to as a basic element in an expression.

7.4.8.4 **RETURN Statement** – The RETURN statement consists of the text

RETURN

This statement returns control from a subprogram (subroutine or function) to the calling program. Normally, the last statement executed in a subprogram is a RETURN statement. Any number of RETURN statements can appear in a subprogram.

7.4.8.5 **BLOCK DATA Statement** – The BLOCK DATA statement is used to establish a block data subprogram, which is a data specification subprogram used to enter initial values for variables in common blocks. No executable statements can appear in a block data subprogram. A block data subprogram is established by a BLOCK DATA statement consisting of the text

BLOCK DATA

This statement declares that the program that follows is a data specification subprogram and must be the first statement of the subprogram. For example,

```
BLOCK DATA
COMMON A,B,C
COMMON/X/ARRAY(100)
INTEGER A,C
REAL B
DATA A,B,C/5,1.5,0/
LOGICAL ARRAY/100*0/
END
```

This subprogram causes blank common to be initialized with the integer 5, a real variable 1.5, an integer 0 as its first three variables and an array with 100 zeros.

The subprogram contains only type statements, EQUIVALENCE, DATA, DIMENSION, and COMMON arguments. A complete set of specifications must be given for an entire common block. A single block data subprogram can initialize any number of named COMMON blocks.

7.4.8.6 EXTERNAL Statement – Function and subroutine subprogram names can be used as the actual arguments of subprograms. When they are, their names must be distinguished from ordinary variables by their appearance in an EXTERNAL statement.

Form EXTERNAL identifier, identifier, . . . , identifier

where identifier is the name of a subprogram.

Effect The identifier is declared a subprogram name and may be used as the argument of other subprograms.

Any subprogram name given as an argument to another subprogram must have previously appeared in an external declaration in the calling program (that is, as an identifier in an EXTERNAL). Example:

```
EXTERNAL SIGMA,THETA
.
.
CALL TRIGF(SIGMA,1.5,ANSWER)
.
.
CALL TRIGF(THETA,187,ANSWER)
.
.
END
SUBROUTINE TRIGF(FUNC,ARG,ANSWER)
.
.
ANSWER=FUNC(ARG)
.
.
RETURN
END
```

7.4.9 OS/78 FORTRAN IV Statement Summary

A summary of all OS/78 FORTRAN IV statements is given in Table 7-16.

Table 7-16 FORTRAN IV Statement Summary

Statement	Form	Effect
Arithmetic	a=b	The value of expression b is assigned to the variable a.
Arithmetic statement function definition	t name(a1...)=x	The value of expression x is assigned to f(a1...) after parameter substitution.
ASSIGN	ASSIGN n TO v	Statement number n is assigned as the value of integer variable v for use in assigned GO TO statement.

Continued on next page

Table 7-16 (Cont.) FORTRAN IV Statement Summary

Statement	Form	Effect
BACKSPACE	BACKSPACE u	Peripheral device u is backspaced one record.
BLOCK DATA	BLOCK DATA	Identifies a block data subprogram.
CALL	CALL prog CALL prog(a1...)	Invokes subroutine named prog, supplying arguments when required.
COMMON	COMMON/block1/a,b,c/...	Variables (a,b,c) are assigned to a common block.
CONTINUE	CONTINUE	No processing, target for transfers.
DATA	DATA var listi/val list1/...	Assigns initial or constant values to variables.
DEFINE FILE	DEFINE FILE a1(b1,c1,U,v1)...	Declares a mass storage file for direct access I/O.
DIMENSION	DIMENSION array (v1..., v7)...	Storage allocated according to dimensions specified for the array.
DO	DO n i=m1,m2,m3	Statements following the DO up to statement n are repeated for values of integer variable i, starting at i=mi, incrementing by m3, terminating when i>m2.
END	END	Cease program compilation; equivalent to STOP in main program or RETURN in subprogram.
END FILE	END FILE u	Writes END-OF-FILE character in file u.
EQUIVALENCE	EQUIVALENCE(v1,v2,...,),	Declares same storage location for variables within parentheses.
EXTERNAL	EXTERNAL subprog, ...	Declares subprograms for use by other subprograms.
FORMAT	FORMAT(spec1,spec2,.../...)	Specifies conversions between internal and external representation of data.
FUNCTION	FUNCTION name(a1...)	Indicates an external function definition.
GO TO		Transfers control to:
	(1) GO TO n	(1) statement n
	(2) GO TO (n1,...,nk),e	(2) to statement n1 if e=1: to statement nk if e=k.

Continued on next page

Table 7-16 (Cont.) FORTRAN IV Statement Summary

Statement	Form	Effect
	(3) GO TO v GO TO v(ni,...,nk) GO TO v,(ni,...,nk)	(3) Transfers control to statement number assigned to v optionally checking that v is assigned one of the labels n1...nk.
IF	IF (arith expr)n1,n2,n3	Transfers control to n1 if expr <0, n2 if expr=0, n3 if expr >0.
IF	IF (logical expr) statement	Executes statement if expression has value .TRUE., otherwise executes next statement in sequence.
Logical	V=E	Value of expression E is assigned to variable V.
PAUSE	PAUSE PAUSE number	Program execution interrupted and number displayed, if given.
READ	READ (u,f) list READ (u,f) READ (u) list READ (a'r) list	Reads a record from a peripheral device according to specifications given in the arguments of the statement.
RETURN	RETURN	Returns control from a subprogram to the calling program.
REWIND	REWIND u	Repositions designated unit to the beginning of the file.
STOP	STOP	Terminate program execution.
SUBROUTINE	SUBROUTINE name(a1,...)	Declares name to be a subroutine and a1,..., if supplied, as dummy arguments.
Type Declaration	type v1,v2,v3,...	Where the variables vn are assigned the indicated type, that is, REAL, INTEGER or LOGICAL.
WRITE	WRITE (u,f) list WRITE (u,f) WRITE (u) list WRITE (u) WRITE (a'r) list	Writes a record to a peripheral device according to specifications given in the arguments of the statement.

CHAPTER 8
BATCH

OS/78 BATCH processing is ideally suited to lengthy or after-hour jobs consisting of fixed sequences of OS/78 system commands not requiring operator intervention. BATCH permits the preparation of a job as a file or part of a file that can be left for an operator to start and run. Output is returned usually in the form of line printer listings and/or diskette files that include program output as well as a comprehensive summary (log) of all action taken by the program, OS/78 BATCH, and the operator.

BATCH provides optional spooling of output files. This allows output to be diverted to fast devices such as diskettes instead of slower devices such as line printers for later transferral to the slower device. This feature serves to increase perceived throughput on a system. A line printer, although optional, makes the use of BATCH more effective. BATCH will support only input/output devices that are present in the hardware configuration.

With a few exceptions, BATCH executes standard OS/78 commands.

8.1 BATCH PROCESSING UNDER OS/78

OS/78 BATCH maintains an input file and an output log. The BATCH input file is a diskette file consisting of a series of BATCH commands. The input file must reside on the system device. Its default extension is .BI. Each command in the BATCH input file generally occupies one line.

The BATCH output file is a line printer listing (log) on which BATCH prints job headers, certain messages that result from conditions within the input file, an image of each line in the input file, and certain types of user output. BATCH supports only the LA78 line printer for the output of the BATCH log. Listings, however, can be output on the LQP78 line printer. If a line printer is not present in the system, the output file is displayed on the terminal.

BATCH accepts program and data files from any input device in the system. User output files may be directed to any output device in the system.

BATCH also permits optional spooling of output files. When spooling is requested, every output request to a non-file-structured device output file is assigned a file name from a list of names maintained by BATCH and directed to a file-structured spool device instead of the user specified device. Spooling of output files increases BATCH throughput, permitting slow output operations to be postponed until a more favorable time. For example, a batch processing run that generates many output listings may be initialized to reroute all listings to a specified diskette. The listings on the diskette may then be dumped onto the appropriate hard copy device after the run, when more time is available. The spool device may be either RXA0, RXA1, SYS: or DSK:.

BATCH is called via the SUBMIT command. The format for a BATCH command string is

```
.SUBMIT SPDV:<DEV:INPUT/option/option
```

where SPDV: is the device on which to spool output. If "SPDV:<" is not specified, no spooling is performed. Note that spooling applies only to non-file-structured output. DEV:INPUT is the input device and file. The default extension for BATCH input files is .BI. The default input device is DSK. The Run-Time Options and their meanings are listed in Table 8-1.

Table 8-1 BATCH Run-Time Options

Option	Meaning
/E (Error option)	Treat Monitor errors as non-fatal errors. If /E is not specified, Monitor errors cause the current BATCH job to be terminated.
/H (Hush option)	Do not output a BATCH log.
/Q (Quiet option)	Output an abbreviated BATCH log, consisting of $JOB and $MSG lines.
/T (Terminal option)	Output the BATCH log to the terminal. This option need be specified only if a line printer is available. If a line printer is not available, the BATCH log is automatically output to the terminal.
/U (Unattended mode)	BATCH will not pause for operator response to $MSG lines. Any attempt to use TTY: as an input device to an unattended BATCH stream will cause the current job to be aborted.
/V (Version option)	Print the version number of OS/78 BATCH on the terminal.

8.2 BATCH MONITOR COMMANDS

A BATCH command is a character or string of characters that begins with the first character of a line in the BATCH input file. Each BATCH command must be followed by a RETURN. The files may contain form feed characters, but form feed characters are ignored by BATCH on input.

BATCH recognizes four monitor level commands. These commands allow routine housekeeping operations in a multi-job, batch processing environment and provide communication between the BATCH user and the operator. Table 8-2 lists the BATCH monitor commands, which may be considered as an extension of the OS/78 Monitor command set. Note that the first character of the $JOB, $MSG and $END commands is a dollar sign (SHIFT/4).

Any record that begins with a dollar sign character but is not one of the BATCH monitor commands listed above is copied onto the output file and ignored by BATCH.

A BATCH processing job consists of a $JOB command line and all of the commands that follow it up to the next $JOB or $END command. Normally, all the commands submitted by one user are processed as a single job, and all output from these commands appears under one job header.

After BATCH encounters a $JOB command, it scans the input file until the next OS/78 command is read. Any lines that follow the $JOB command and precede the first command are written onto the log and ignored by BATCH.

The first character of every command is a dot (.). An exception to this is the use of an asterisk (*) whose function is to accept a command string that indicates input/output files. It is further described in the example program given in Section 8.3. The rest of the line contains a command, which should appear in

Table 8-2 BATCH Monitor Commands

Command	Meaning
$JOB	Initialize for a new job and display a job header on the output file. The remainder of the $JOB record is included in the job header but ignored by BATCH. It should be used for job identification, to provide correlation between terminal output, line printer output and spool device output.
$MSG	Sounds the terminal buzzer and prints an image of the message at the terminal. If the /U option was not specified, implying that an operator is present, BATCH will pause until any key is struck at the keyboard. If the /U option was specified, processing continues uninterrupted.
$END	Terminate batch processing and exit to the Monitor. A $END command should be the last line of every BATCH input file.
/	Copy the line onto the output log, then ignore it. BATCH assumes that every line beginning with a slash is a comment.

standard OS/78 format. However, commands that would be terminated with an ESCape under interactive OS/78 should be terminated with a dollar sign under BATCH. Every standard command is legal input to BATCH except the MEMORY command. However, the ODT command will go to the terminal for input instead of the BATCH file. It is not usually meaningful to invoke ODT, EDIT, or other interactive programs under BATCH.

BATCH executes an OS/78 command by stripping off the initial dot character and passing the remainder of the line to the Monitor. BATCH then passes control to the Monitor, which executes the command as though it had been typed at the keyboard. Monitor commands that return control to the Monitor level should be followed by a BATCH monitor command or another Monitor command. The SUBMIT command can be used to chain from one BATCH stream to another, if desired.

The general rules and conventions associated with BATCH processing are as follows:

1. The dollar sign ($) is always in the first character position of the BATCH command lines $JOB, $MSG and $END.
2. Each job must have a $JOB and $END command.
3. OS/78 commands can be spelled out entirely or in accordance with the accepted abbreviations. Also, a dot (.) must precede every command.
4. Wild cards can be specified only for the OS/78 commands COPY, DELETE, DIRECT, LIST, RENAME and TYPE.
5. Comments may only be included as separate comment lines.
6. Only 80 characters per control statement are allowed.
7. Continuation file specification lines (where necessary) are specified by an initial asterisk. For example,

```
.LOAD PROG,SUBA,SUBB$
*SUBC,SUBD,SUBE$
```

8.3 THE BATCH INPUT FILE

The following file is an example of a Batch input file.

```
$JOB
.DATE 3-JUN-77
/LIST ANY HELP FILES AND STARTING BLOCKS
.DIR *.HL/B
$MSG INSERT DISKETTE INTO DRIVE 1 - TYPE ANY KEY TO CONTINUE
/LIST DIRECTORY OF DISKETTE 2 - SPOOL TO FILE BTCHA1
.DIR TTY:<RXA1:
/COPY FILE FROM DISKETTE 2 DISKETTE 1
.COPY RXA0:<RXA1:POWER.FT
/COMPILE FORTRAN PROGRAM POWER.FT
.COMPILE POWER.FT
/LOAD FORTRAN PROGRAM AND CHAIN TO RUN TIME SYSTEM
.LOAD POWER.RL/G$
/STORE RESULTS OF FORTRAN PROGRAM IN FILE 'HOLD'
*HOLD.TM</4$
/EXECUTE FORTRAN PROGRAM AND DISPLAY RESULTS ON SCREEN
.EXE POWER.LD
/END OF JOB NO.1 - START NEXT JOB
$JOB
/ASSEMBLE FILE SAMPLE AND PRODUCE CREF LISTING
.PAL SAMPLE/C-LS
/EXECUTE PROGRAM SAMPLE
.EXE SAMPLE.BN
/END OF EXAMPLE AND BATCH INPUT FILE
$END
```

The file was created using the Editor and is named BATSAM.BI. For the example used, both the PAL8 and FORTRAN IV system programs were on diskette RXA0.

BATCH is started using the SUBMIT command and typing

```
.SUBMIT RXA0:<BATSAM
```

when spooling is desired, or

```
.SUBMIT BATSAM
```

when spooling is not desired. The default extension for the Batch input file is .BI.

BATCH begins processing by printing a job header and executing the DATE command. BATCH next executes the DIRECT command which displays any Help file names and their starting blocks. Wild cards are used in this command.

BATCH continues to scan the input file for the next line. BATCH processes the $MSG command, sounds the terminal buzzer, and copies the $MSG record onto the terminal. Assuming that an operator is present, processing is suspended until any key is typed at the terminal.

The operator performs the task specified by the $MSG command that of placing diskette 2 into Drive 1, and types any key, allowing BATCH to continue reading the input file.

BATCH then executes the DIRECT command which lists the directory of RXA1 on the line printer. Since spooling is active because RXA0 was specified as the spooling device in the SUBMIT command line and a non-file-structured device is specified as output in the DIRECT command, BATCH intercepts this output and stores it in a temporary file on the spool device. The output is stored in a file named BTCHA1. BATCH then outputs the message

SPOOL TO FILE BTCHA1

on both the console terminal and the line printer, if available. If another file is routed to the spool device it will be assigned the file name BTCHA2, and any successive files will be named in the following sequence:

```
BTCHA3
  .
  .
  .
BTCHA9
BTCHB0
  .
  .
  .
BTCHZ9
```

A total of 260 spool device files is permitted. If output to a spool device file is generated by a program that appends a default extension to output file names, the spool device file will be assigned a standard default extension. All of the spool device files may then be transferred to the terminal or line printer by using the TYPE or LIST command with the input file specification dev:BTCH??.*.

You may type CTRL/C at any time during a batch processing run. Typing CTRL/C at the program level causes an effective jump to location 07600, which recalls the BATCH monitor. The BATCH monitor then recognizes the CTRL/C and terminates the BATCH run. Sometimes two CTRL/C's are required to be typed in succession to return to the Monitor dot.

Continuing with our example program, the next command copies the FORTRAN program POWER.FT from Drive 1 to Drive 0. The program then is compiled creating a module POWER.RL that is used by the LOAD command to generate a loader image file. The LOAD command is given the /G option that chains to the run-time system for program execution.

Note that the ESCape function is specified via a dollar sign ($) to call a system program called the Command Decoder. This program (which normally displays an asterisk when it is running) allows the run-time system to accept file input/output specifications. Similarly, an asterisk is used for specifying input/output files. In this case, a file HOLD.TM is designated to store the output of the executed FORTRAN program. The file specification line is then followed by a dollar sign (again serving as an ESCape key function) to execute the LOAD command. After the execution of this command, the loader image file POWER.LD has been created, and its executable results stored in the HOLD.TM file. The EXECUTE command executes the program POWER.FT, displaying the results on the terminal.

BATCH then encounters the second $JOB command. The first job is terminated and a new header is printed. The second job calls CREF to assemble a source program named SAMPLE and produce a CREF listing. The results of this command is to produce a binary file called SAMPLE.BN and a CREF listing stored in the file

SAMPLE.LS. Typing the TYPE or LIST commands will print this file on the terminal or line printer, respectively. The program is then executed. The BATCH job is terminated upon encountering the $END command and control returns to the Monitor.

The BATCH $END command always must appear as the last record in the input file to terminate batch processing and cause BATCH to recall the Monitor and re-establish interactive processing under OS/78.

8.4 BATCH ERROR MESSAGES

BATCH generates two types of error messages. They are BATCH error messages and system error messages. BATCH level error messages appear in the form:

#BATCH ERR

System error messages are generated by OS/78 system programs. When these occur, BATCH will append a "#" character to the beginning of the message, so that it appears in the form:

#SYSTEM ERROR

Any occurrence of an error normally causes BATCH to terminate the current job and scan the input file for the next $JOB command. If the /E option was specified, BATCH treats errors as non-fatal and continues the BATCH run.

Table 8-3 lists the BATCH error messages, their meanings, and the probable cause for the error.

Table 8-3 BATCH Error Messages

BATCH Error Message	Meaning
#MONITOR OVERLAYED	The BATCH monitor was called to accept and transmit a file specification, but found that a user program had overlayed part or all of the BATCH monitor. BATCH then will execute the next command.
#BAD LINE JOB ABORTED	The BATCH monitor detected a line in the input file that did not have one of the characters dot, slash, dollar sign or asterisk as the first character of the record. The record is ignored, and BATCH scans the input file for the next $JOB command.
#SPOOL TO FILE BTCHA1	Where the "A" may be any character of the alphabet and the "1" may be any decimal digit. This message indicates that BATCH has intercepted a non-file-structured output file and routed it to the spool device. This is not, generally, an error condition. Spool device file names are assigned sequentially, beginning with file BTCHA1. Standard default extensions may be assigned by some system programs.
#MANUAL HELP NEEDED	BATCH is attempting to operate an I/O device, such as TTY, that will require operator intervention. If the /U option was specified to indicate that an operator is not present, this message is suppressed, the current job is terminated, and BATCH scans the input file for the next $JOB command record. If an operator is present, the $MSG command provides notification of the action that should be taken.

Continued on next page

Table 8-3 (Cont.) BATCH Error Messages

BATCH Error Message	Meaning
#ILLEGAL INPUT	A file specification designated TTY as an input device when the /U option indicated that an operator is not available. The current job is terminated, and BATCH scans the input file for the next $JOB command.
#INPUT FAILURE	Either a hardware problem prevented BATCH from reading the next line of the input file, or BATCH read the last line of the input file without encountering a $END command line. If a hardware problem exists, correct the problem and type any character at the terminal to resume processing.
#SYS ERROR	A hardware problem prevented BATCH from performing an I/O operation. Program execution halts, and the system must be restarted, using the START pushbutton.
BATCH.SV NOT FOUND	A copy of BATCH.SV must exist on the system device. Control returns to the Monitor.
DEV NOT IMPLEMENTED	BATCH cannot accept input from the specified input device because its handler is not permanently resident. Only input from SYS: is permitted. Control returns to the Command Decoder (See Appendix D). In most cases, type CTRL/C and then retype the command, using the correct parameters.
ILLEGAL SPOOL DEVICE	The device specified as a spooling output device must be file-structured. Control returns to the Command Decoder (See Appendix D). In most cases, type CTRL/C and then retype the command, using the correct parameters.
%BATCH SQUISHING SYS:!	Batch is running and attempting to squish the system which may cause BATCH.SV or the Batch input file to be moved.

8.5 RESTRICTIONS UNDER OS/78 BATCH

OS/78 BATCH is unprotected from user errors. The BATCH monitor resides in locations 5000 to 7577 in Field 3. BATCH also uses the following locations in Field 0 and Field 3.

LOCATION	USED AS:
07777	Batch processing flag.
37774-37777	Internal pointers.

Both the Monitor and the Command Decoder (See Appendix D) check the batch processing flag (bit 1 of 07777) whenever they are entered from the program level. Any user program that modifies location 07777 may cause batch processing to be terminated prematurely before the next line of the BATCH input file is read unless bit 1 is left alone.

When the Monitor is entered from the Program level (JMP to 07600 or 07605) it checks the batch processing flag and reads a new copy of the BATCH monitor into memory if batch processing is in progress. The Command Decoder, however, does NOT perform this operation. Thus, the Command Decoder must not be called unless the BATCH monitor is already in memory.

Therefore, large user programs may be loaded over the BATCH monitor as long as they do not modify the last four locations in Field 3. However, once a user memory load has overwritten the BATCH monitor, execution must remain at the program level until the Monitor has been re-entered and a new copy of the BATCH monitor is read into memory. The Command Decoder must not be called after a user program has been loaded over the BATCH monitor.

In general, this restriction applies only to loader programs and only when the loader calls the Command Decoder more than once while building a large memory load. Multiple calls to the Command Decoder may be avoided when loading large programs during batch processing if the memory load is first built in a stand-alone environment and then saved for subsequent execution under BATCH.

In conjunction with this, note that it is impossible to save the memory image of any program that overlays the BATCH monitor, under BATCH. After the load operation but before the save is executed, the BATCH monitor will be read back into memory, destroying part of the user program. Thus, the Monitor SAVE operation will cause part of the BATCH monitor to be saved instead of that part of the user program which originally overlayed the BATCH monitor.

A BATCH job must never move or delete either BATCH.SV or the BATCH input file. Also, SYS: should never be SQUISHed while BATCH is running because it may cause these files to move.

CHAPTER 9
DEBUGGING A PROGRAM

OCTAL DEBUGGING TECHNIQUE (ODT)

ODT is an interactive system program that makes debugging PAL8 programs easy by providing selective execution and memory examination, modification, and searching facilities.

9.1 ODT FEATURES

ODT features include examination and modification of memory locations, and the use of instruction breakpoints to return control to ODT. ODT makes no use of the program interrupt facility and, with few restrictions, is invisible to a user program.

The breakpoint is one of ODT's most useful features. When debugging a program, it is often desirable to allow the program to run normally up to a predetermined point at which you may examine and possibly modify the contents of the accumulator (AC), the link (L), or various instructions or storage locations within the program, depending on the results found. To accomplish this, ODT acts as a monitor to the user program.

You decide how far the program is to run and instruct ODT to insert an instruction in the program which, when encountered, causes control to transfer back to ODT. ODT immediately preserves in internal storage locations the contents of the AC and L at the breakpoint. It then displays the location at which the breakpoint occurred, as well as the contents of the AC at that point. ODT will allow examination and modification of any location of the program (or those ODT locations containing the AC and L). The breakpoint may also be moved or deleted, and a request made that ODT continue running the program. This causes ODT to restore the AC and L, execute the "trapped" instruction and continue in the program until the breakpoint is again encountered or the program is terminated.

9.2 CALLING AND USING ODT

When ODT is being used, a complete assembly listing of the program should be available for the program that is being debugged.

Before calling ODT, place the program that is to be debugged in memory as the "current program". For example, if it is a memory image file, type

```
.GET SYS SAMPLE

.ODT
```

If it is a binary file, type

```
.LOAD SAMPLE

.ODT
```

Little of the memory that is being used is disturbed by the running of ODT, because the sections of the program which ODT may occupy when in memory are preserved on the system device and swapped back into memory as necessary. ODT uses the Job Status Word of the particular program to determine whether or not swapping should occur. If the program does not use locations 0-1777 in field 0, less swapping occurs during use of the breakpoint feature.

If any amount of a program is typed directly into memory (in octal), the Core Control Block of the program may not reflect the true extent of the program. If octal additions are made below location 2000 in field 0, ODT may give erroneous results. This condition is corrected by changing the Job Status Word, which is stored in location 7746 of field 0, and which can be examined and changed using ODT as explained later in this chapter. Location 7745 of field 0 is the 12-bit starting address of the program in memory and location 7744 contains the field designation in the form 62n3, where n is the field designation of the starting address.

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

1. If a breakpoint is inserted at a location which contains an auto-indexed instruction, the auto-index register is incremented immediately after the breakpoint. Thus, when control returns to the user in ODT, the register will have been increased by one. The breakpoint instruction is executed properly, but the index register, if examined, will be one greater than it should.
2. ODT keeps track of the terminal display I/O flag and restores the terminal flag when it continues from a breakpoint.
3. The breakpoint feature uses and does not restore locations 4, 5, and 6 in the memory field in which the breakpoint is set.
4. The breakpoint feature of ODT uses the table of user-defined device names as scratch storage, destroying any device names that may have created and creating garbage entries. Therefore, it is advisable not to use user-defined device names in programs being debugged with ODT breakpoints. After a session with ODT in which breakpoints are used, give a DEASSIGN command to clear out the user-device name table.
5. Breakpoints must not be set in the Monitor, in the device handlers, or between a CIF and the following JMP or JMS instruction.
6. ODT should not be used to debug programs which use interrupts.

If any operations are attempted in non-existent memory, ODT ignores the command and types "?". Thus, attempting to examine locations in field 4 and above, ODT responds with ?.

Typing CRTL/C returns control to the Keyboard Monitor; the program can be saved on any file-oriented device, using the SAVE command. If this is done, no other system commands should be given before the SAVE command.

9.3 ODT COMMANDS

9.3.1 Special Characters

These characters will be illustrated using the SAMPLE.PA program created in Chapter 4.

9.3.1.1 Slash(/) – Open This Location – The location examination character (/) causes the location addressed by the octal number preceding the slash to be "opened" and its contents displayed in octal. The open location can then be modified by typing the desired octal number and closing the location by using the RETURN key. If a number which is not octal or the DELETE key is typed, a question mark (?) is displayed and the number is ignored. Any octal number from 1 to 4 digits in length is legal input. If more than 4 digits are entered, only the last 4 entered are accepted by ODT. Typing / with no preceding argument opens the most recently opened location. For example,

```
200/7300
201/6046
201/6046 6048?
201/6046 2345
/2345
```

The above example modifies locations in Field 0. If locations are examined in Fields 1, 2, or 3, specify the location as fnnnn/ where f is the field and nnnn is the location in octal.

9.3.1.2 RETURN – Close Location – If you have typed a valid octal number after the content of a location is displayed by ODT, pressing the RETURN key causes that number to replace the original contents of the opened location and the location to be closed. If nothing has been typed by you, the location is closed but the contents of the location are not changed. For example,

```
201/6046            location 201 is unchanged.
201/6046  2345      location 201 is changed to contain 2345.
/2345  6046         place 6046 back in location 201.
```

Typing another command will also close an opened register.

9.3.1.3 LINE FEED – Close Location, Open Next Location – The LINE FEED key has the same effect as the RETURN key, but, in addition, the next sequential location is opened and its contents displayed. For example,

```
200/7300 (LF)              location 200 is closed unchanged and 201 is opened. Type change.
00201  /2346 6046 (LF)     201 is closed (containing 6046) and 202 is opened.
00202  /1216
```

Semicolon(;) – Deposits the octal value typed (if any) in the currently opened location, closes that location and opens the next sequential location for modification. For example,

```
202/1216  1234;5670
202/1234
203/5670
```

A series of octal values can be deposited sequentially using the semicolon character. Typing multiple semicolons skips a memory location for each semicolon typed and will accept an octal value at the location skipped to. For example,

```
202/1234  1216;;;0000
202/1216
203/5670
204/6041
205/0000
```

n+ and n– – Opens the current location plus n or minus n for modification and prints the content of that location. If n is omitted, it is assumed to be 1. For example,

```
200/7300  3+
00203/5670  3217
```

or

```
207/6046  2-
00205/0000  5204
```

9.3.1.4 Circumflex (^) – Close Location, Take Contents as Memory Reference Instruction and Open Effective Address – The circumflex will close an open location just as will the RETURN key. Further, it will interpret the contents of the location as a memory reference instruction, open the location referenced and display its contents. For example,

`202/1216 ^` `00216 /0220`	1216 symbolically is "TAD, this page, relative location 16," so ODT opens location 216.

The indirect bit is used in determining the effective address.

9.3.1.5 Underline (_) – Close Location, Open Indirectly – The back arrow will close the currently open location and then interpret its contents as the address of the location (in the same field) whose contents it is to print and open for modification. For example,

```
202/1216 ^
00216 /0220 _
00220 /0215
```

9.3.2 Illegal Characters

Any character that is neither a valid control character nor an octal digit causes the current line to be ignored and a question mark printed. For example,

`2:?` `2U?`	ODT opens no location.
`206/1617 67K?` `/1617`	ODT ignores a partial number and closes location 206.

9.3.3 Control Commands

9.3.3.1 fnnnnG – Transfer Control to Program at Location nnnn of field f – Clear the AC and L then go to the location fnnnn. The breakpoint, if any, will be inserted. Typing G alone will cause a start at location 0000.

9.3.3.2 fnnnnB – Set Breakpoint at Location nnnn of field f – Instructs ODT to establish a breakpoint at the location fnnnn. A breakpoint may be changed to another location whenever ODT is in control, by simply typing fnnnnB where fnnnn is the new location. Only one breakpoint may be in effect at a time; therefore, requesting a new breakpoint removes any previously existing one. A breakpoint may be established at location 00000.

The breakpoint (B) command does not make the actual exchange of ODT instruction for user instruction, it only sets up the mechanism for doing so. The actual exchange occurs when a G or a C command is executed.

When, during execution, the program encounters the location containing the breakpoint, control passes immediately to ODT (via locations 0004-0006 of the instruction field). The contents of the accumulator and contents of the link at the point of the interruption are saved in special locations accessible to ODT. The user instruction that the breakpoint was replacing is restored, before the address of the trap and the content of the AC are displayed. The restored instruction has not been executed at this time, and will not be executed until the "continue from breakpoint" command is given. Any location used, including those containing the stored accumulator and Link, can now be modified. The breakpoint can also be moved or removed at this time.

9.3.3.3 B – Remove Breakpoint – Typing B alone removes any previously established breakpoint and restores the actual contents of the breakpoint location. B should be typed before saving the current program if a breakpoint was set. Typing B where no breakpoint is set has no effect.

NOTE

If a breakpoint set by ODT is not encountered while ODT is running the object (user's) program, the instruction which causes the break to occur will not be removed from the user's program.

9.3.3.4 A – Open C(AC) Location – When the breakpoint is encountered the contents of the accumulator and the contents of the link are saved for later restoration. Typing A after having encountered a breakpoint opens for modification the location in which the AC was saved and prints its contents. This location may now be modified in the normal manner (see Slash) and the modification will be restored to the accumulator when the C or G commands are given (contains either 0000 or 0001).

9.3.3.5 L – Open C(L) Location – Typing L opens the link storage location for modification and prints its contents. The link location may now be modified as usual (see Slash) and that modification will be restored to the Link when the C or G commands are given.

9.3.3.6 C – Continue From a Breakpoint – Typing C, after having encountered a breakpoint, causes ODT to restore the contents of the accumulator, link, and breakpoint location, and transfer control to the breakpoint location. The user program then runs until the breakpoint is again encountered or the program halts or exits to the Monitor.

9.3.3.7 nnnnC – Continue nnnn +1 Times from Breakpoint – A breakpoint may be established at some location within a loop of a program. Since loops often run to many iterations, some means must be available to prevent a break from occurring each time the break location is encountered. This is the function of nnnnC (where nnnn is an octal number). The "continue" operation is done nnnn +1 times. If nnnn is omitted, 0 is assumed. Thus, 2C will allow a loop in which the breakpoint is embedded to execute three times.

Given the following program ADD1, which increases the value of the accumulator by increments of 1, the use of the Breakpoint and Continue commands may be illustrated.

```
*200

        0200  *200
00200   7300          CLA CLL
00201   1206  A,      TAD ONE
00202   2207  B,      ISZ CNT
00203   5202          JMP B
00204   5201          JMP A
00205   7402          HLT
00206   0001  ONE,    1
00207   0000  CNT,    0
```

Assemble the program and call ODT by typing

```
.PAL ADD1/L

.ODT
```

```
00201B
00200G
00201   (0;0000
C
00201   (0;0001
C
00201   (0;0002
4C
00201   (0;0006
```

ODT has been loaded and started. A breakpoint is inserted at location 0201 and execution stops here showing the accumulator initially set to 0000. The use of the Continue command (C) executes the program until the breakpoint is again encountered (after one complete loop) and shows the accumulator to contain a value of 0001. Again execution continues, incrementing the AC to 0002. At this point, the command 4C is used, allowing execution of the loop to continue five more times before stopping at the breakpoint. The contents of the accumulator have now been incremented to 0006.

9.3.3.8 D – Open Data Field – Typing D opens for modification the internal ODT location containing the data field which was in effect at the last breakpoint. Contents of D always appear as multiples of 10, that is, 10 means field 1, 20 means field 2.

9.3.3.9 F – Open Current Field – Typing F opens for modification the internal ODT location containing the field used by ODT in the W (search) command, in the _ and ^ (indirect addressing) commands, or in the last breakpoint (depending upon which was used most recently). The contents of F are always expressed as multiples of 10 (as in the D command).

9.3.3.10 M – Open Search Mask – Typing M causes ODT to open for modification the internal ODT location containing the current value of the search mask and print its contents. Initially the mask is set to 7777. It may be changed by opening the mask location and typing the desired value after the value printed by ODT, then closing the location.

9.3.3.11 M (LF) – Open Lower Search Limit – The word immediately following the mask storage location contains the location at which the search is to begin. Pressing the LINE FEED key to close the mask location causes the lower search limit to be opened for modification and its contents displayed. Initially the lower search limit is set to 0000. It may be changed by typing the desired lower limit after that printed by ODT, then closing the location.

9.3.3.12 M (LF) (LF) – Open Upper Search Limit – The next sequential word contains the location with which the search is to terminate. Pressing the LINE FEED key to close the lower search limit causes the upper search limit to be opened for modification and its contents printed. Initially, the upper search limit is 7577. It may be changed by typing the desired upper search limit after the one printed by ODT, then closing the location with the RETURN key.

9.3.3.13 nnnnW – Word Search – The command nnnnW (where nnnn is an octal number) will cause ODT to conduct a search of a defined section of memory, using the mask and the lower and upper limits that the user has specified, as indicated above. The word searching operation determines if a given quantity is present in any of the locations of that defined section of memory. A search never alters the contents of any location.

The search is conducted as follows: ODT masks each location within the limits specified and compares the result to the quantity for which it is searching. If the two quantities are identical, the address and the actual unmasked contents of the matching location are displayed and the search continues until the upper limit is reached. The search includes the upper limit.

For example, locations 3000 through 3777 are to be searched for all ISZ instructions, regardless of what location they refer to (that is, search for all locations beginning with an octal 2 which is the operation code for an ISZ instruction).

M/7777 7000 (LF)	Change the mask to 7000, open lower search limit.
0041/5273 3000 (LF)	Change the lower limit to 3000, open upper limit.
0042/1335 3777	Change the upper limit to 3777, close location.
2000W	Initiate the search for ISZ instructions.
00005 /2331	These are four ISZ instructions in this section of memory. Note that these might also be data values that happen to start with an octal "2".
00006 /2324	
00011 /2222	
00033 /2575	

9.4 ERRORS

The only legal inputs are command characters and octal digits. Any other character will cause the character or line to be ignored and a question mark to be printed by ODT. When G is used, it must be preceded by an address to which control will be transferred. If not preceded by an address, a question mark will not be displayed, but control will be transferred to location 0.

9.5 PROGRAMMING NOTES

ODT will not turn on the program interrupt. It does, however, turn off the interrupt when a breakpoint is encountered, to prevent spurious interrupts.

Breakpoints are fully invisible to "open location" commands; however, breakpoints must not be placed in locations which the user program will modify in the course of execution or the breakpoint may be destroyed. Caution should be used in placing a breakpoint between a call to USR function code 10 (USRIN) and the following call to USR function code 11 (USROUT).

If a trap set by ODT is not encountered by your program, the breakpoint instruction will not be removed. If the SAVE command is to be used, the B command must be used to remove the breakpoint before typing CTRL/C. ODT should not be run under BATCH.

9.6 ODT COMMAND SUMMARY

Table 9-1 presents a brief summary of the ODT commands. All addresses can be input as 5 digits, and are printed as 5 digits.

Table 9-1 ODT Command Summary

Command	Meaning
fnnnn/	Open location designated by the octal number fnnnn, where the first digit represents the memory field. ODT displays the contents of the location, a space, and waits for the user to enter a new value for that location or close the location. If f is omitted, field 0 is assumed.
/	Reopen latest opened location.
nnnn;	Deposit nnnn in the currently opened location, close that location and open the next sequential location for modification. A series of octal values can be deposited in sequential locations through use of the ; character. Multiple ;'s skip a memory location for each ; typed and prepare to insert subsequent values beyond the one(s) skipped.
RETURN key	Close the currently open location, if any.
LINE FEED key	Close location; open the next sequential location for modification, and display the contents of that location.
n+	Open the location n after the current location for modification and display the contents of that location.
n-	Open the location n before the current location for modification and display its contents.
^ (Circumflex)	Close current location, take contents of the current location as a memory reference instruction and open the location referenced, printing its contents. **NOTE** No distinction is made between instruction op-codes when using ^. Thus, all op-codes (0-7) are treated as memory reference instructions. Also, great care should be exercised when using ^ with indirectly referenced auto-index registers. If ^ is used in this case, the contents of the auto-index register is incremented by one. Check to see that the register contains the proper value before proceeding (using C or G commands or word searched).
_(Underline)	Close current location, take contents of the current location as a 12-bit address and open that address for modification, displaying its contents.
fnnnnG	Transfer control of program to location fnnnn, where the first digit represents the memory field.
fnnnnB	Establish a breakpoint at location fnnnn, where the first digit represents the memory field. Only one breakpoint is allowed at any given time.

Continued on next page

Table 9-1 (Cont.) ODT Command Summary

Command	Meaning
B	Remove the breakpoint, if any.
A	Open for modification the location in which the contents of the Accumulator were stored when the breakpoint was encountered.
L	Open for modification the location in which the contents of the Link were stored when the breakpoint was encountered.
C	Continue from a breakpoint.
nnnnC	Continue from a breakpoint nnnn+1 times before interrupting the user's program at the breakpoint location.
M	Open the search mask, initially set to 7777, which can be changed by typing a new value.
M (LF)	Open the lower search limit. Type in the location (four octal digits) where the search will begin.
M (LF) (LF)	Open the upper search limit. Type in the location (four octal digits) where the search will terminate.
nnnnW	Search the portion of memory as defined by the upper and lower limits for the octal value nnnn. Search can only be done on a single memory field at a time. See the F command.
D	Open for modification the word containing the data field which was in effect at the last breakpoint. Contents of D always appear as multiples of 10(8), this is, 10 means field 1, 20 field 2.
F	Open for modification the word containing the field used by ODT in the W (search) command, in the ^ and _ (indirect addressing) commands, or in the last breakpoint (depending upon which was used most recently). The contents of F are always expressed as multiples of 10 (octal) (as in the D command).
CTRL/D	Interrupt a long search output and wait for the next ODT command.
DELETE key	Cancels previous number typed, up to the last non-numeric character typed.

APPENDIX A

CHARACTER/CONTROL CODES AND SPECIAL SYMBOLS

Character Codes

8-bit ASCII Code	6-bit Code	Character Representation	Remarks
240	40		space (non-printing)
241	41	!	exclamation point
242	42	"	quotation marks
243	43	#	number sign
244	44	$	dollar sign
245	45	%	percent
246	46	&	ampersand
247	47	'	apostrophe or acute accent
250	50	(	opening parenthesis
251	51	)	closing parenthesis
252	52	*	asterisk
253	53	+	plus
254	54	,	comma
255	55	—	minus sign or hyphen
256	56	.	period or decimal point
257	57	/	slash
260	60	0	
261	61	1	
262	62	2	
263	63	3	
264	64	4	
265	65	5	
266	66	6	
267	67	7	
270	70	8	
271	71	9	
272	72	:	colon
273	73	;	semicolon
274	74	<	less than
275	75	=	equals
276	76	>	greater than
277	77	?	question mark
300	00	@	at sign
301	01	A	
302	02	B	
303	03	C	
304	04	D	

Continued on next page

Character Codes (Cont.)

8-bit ASCII Code	6-bit Code	Character Representation	Remarks
305	05	E	
306	06	F	
307	07	G	
310	10	H	
311	11	I	
312	12	J	
313	13	K	
314	14	L	
315	15	M	
316	16	N	
317	17	O	
320	20	P	
321	21	Q	
322	22	R	
323	23	S	
324	24	T	
325	25	U	
326	26	V	
327	27	W	
330	30	X	
331	31	Y	
332	32	Z	
333	33	[	opening bracket
334	34	\	backslash
335	35	]	closing bracket
336	36	^	circumflex
337	37	_	underline

Control Codes

8-bit ASCII Code	Character Name	Remarks
000	null	Ignored in ASCII input.
203	CTRL/C	Tells program to return to Keyboard Monitor, echoed as ^C.
207	BELL	CTRL/G. Sounds buzzer
211	TAB	CTRL/I, horizontal tabulation.
212	LINE FEED	Used as a control character by the Command Decoder and ODT. Feeds line
213	VT	CTRL/K, vertical tabulation.
214	FORM	CTRL/L, form feed.
215	RETURN	Carriage return, generally echoed as carriage return followed by a line feed, returns to left side of screen on current line.
217	CTRL/0	Break Character, used conventionally to suppress terminal output, echoed as ^0.
225	CTRL/U	Delete current input line, echoed as ^U.
232	CTRL/Z	End-of-File character for all ASCII and binary files (in relocatable binary files CTRL/Z is not a terminator if it occurs before the trailer code).
233	ESC	ESCape replaces and is considered equivalent to ALTMODE. Used in Escape sequences.
377	DELETE	Deletes the previous character typed.

Special Symbols (VT-52)

<table>
<tr><th>7-bit ASCII Code</th><th>Graphics Mode</th><th>Non-Graphics Mode</th></tr>
<tr><td>136</td><td>blank</td><td>^</td></tr>
<tr><td>137</td><td>blank</td><td>_</td></tr>
<tr><td>140</td><td>reserved</td><td>`</td></tr>
<tr><td>141</td><td>solid rectangle</td><td>a</td></tr>
<tr><td>142</td><td>$^{1}/$</td><td>b</td></tr>
<tr><td>143</td><td>$^{3}/$</td><td>c</td></tr>
<tr><td>144</td><td>$^{5}/$</td><td>d</td></tr>
<tr><td>145</td><td>$^{7}/$</td><td>e</td></tr>
<tr><td>146</td><td>degrees</td><td>f</td></tr>
<tr><td>147</td><td>plus or minus</td><td>g</td></tr>
<tr><td>150</td><td>right arrow</td><td>h</td></tr>
<tr><td>151</td><td>ellipsis</td><td>i</td></tr>
<tr><td>152</td><td>divide by</td><td>j</td></tr>
<tr><td>153</td><td>down arrow</td><td>k</td></tr>
<tr><td>154</td><td>bar at scan 0</td><td>l</td></tr>
<tr><td>155</td><td>bar at scan 1</td><td>m</td></tr>
<tr><td>156</td><td>bar at scan 2</td><td>n</td></tr>
<tr><td>157</td><td>bar at scan 3</td><td>o</td></tr>
<tr><td>160</td><td>bar at scan 4</td><td>p</td></tr>
<tr><td>161</td><td>bar at scan 5</td><td>q</td></tr>
<tr><td>162</td><td>bar at scan 6</td><td>r</td></tr>
<tr><td>163</td><td>bar at scan 7</td><td>s</td></tr>
<tr><td>164</td><td>subscript 0</td><td>t</td></tr>
<tr><td>165</td><td>subscript 1</td><td>u</td></tr>
<tr><td>166</td><td>subscript 2</td><td>v</td></tr>
<tr><td>167</td><td>subscript 3</td><td>w</td></tr>
<tr><td>170</td><td>subscript 4</td><td>x</td></tr>
<tr><td>171</td><td>subscript 5</td><td>y</td></tr>
<tr><td>172</td><td>subscript 6</td><td>z</td></tr>
<tr><td>173</td><td>subscript 7</td><td>{</td></tr>
<tr><td>174</td><td>subscript 8</td><td>|</td></tr>
<tr><td>175</td><td>subscript 9</td><td>}</td></tr>
<tr><td>176</td><td>paragraph</td><td>~</td></tr>
</table>

Uses of the Special Symbols

The Graphics mode is entered when codes ESC F (233 306) are received in succession. The Graphics mode is left when codes ESC G (233 307) are received.

The codes 154-163 cause eight horizontal lines at various scans within the character position to be displayed. These bars can be used to paint a bar-graph on the screen with more accuracy than would be possible using only minus signs and underlines, for instance.

The codes 142-145 ("$^{1}/$", "$^{3}/$", "$^{5}/$", and "$^{7}/$") are used preceding the subscripts to form fractions. In particular, the fractions $^{1}/_{8}$, $^{1}/_{4}$, $^{3}/_{8}$, $^{1}/_{2}$, $^{5}/_{8}$, $^{3}/_{4}$, and $^{7}/_{8}$ can be formed using these four symbols and the subscripts.

APPENDIX B
USEFUL MATHEMATICAL SUBROUTINES

Since the DECstation 78 computer system contains no built-in multiplication or division instructions, it is necessary to calculate these functions explicitly in any assembly language programs in which they are needed. The subroutines listed in this appendix illustrate efficient ways to do these calculations. Note that these are single-precision, unsigned calculations. "Single-precision" means that all numbers are represented by 12-bit words, and "unsigned" means that all numbers are considered positive quantities, with no explicit sign bit. This means that numbers range from 0 to 7777 (octal), inclusive. Another important point is that single-precision, unsigned numbers may be regarded as integers or as fractions, depending on whether the binary point (analogous to the decimal point in the decimal number system) is assumed to be located to the right of the right-most bit (in the case of integers) or to the left of the left-most bit (in the case of fractions).

B.1 UNSIGNED INTEGER MULTIPLICATION SUBROUTINE

```
/CALCULATES A*B, WHERE A AND B ARE UNSIGNED INTEGERS. IF A*B>7777
/A JMP IS MADE TO LOCATION 'ERROR'.

/CALLING SEQUENCE:
/       JMS ML
/        A
/        B
/CONTROL RESUMES HERE WITH (I.E., AT THE THIRD LOCATION AFTER THE
/JMS INSTRUCTION) WITH ACCUMULATOR(AC)=A*B. A AND B ARE PRESERVED.
/LINK=0 ON RETURN.

/NOTE: USE REPETITIVE ADDITION INSTEAD OF THIS SUBROUTINE
/IF EITHER ARGUMENT WILL ALWAYS BE LESS THAN ABOUT 50 (OCTAL).

/NOTE: STARRED INSTRUCTIONS MAY BE REMOVED IF THEIR EFFECTS
/ARE NOT NEEDED.

/NOTE: REMOVING ALL OVERFLOW TEST INSTRUCTIONS GIVES A SUBROUTINE
/WHICH TRUNCATES THE RESULT, RETURNING A*B MOD 10000.

ML,     0
        CLA                 /* IGNORE AC
        TAD I ML            /GET MULTIPLIER
        ISZ ML
        DCA ML2
        TAD KM14
        DCA COUNT
ML1,    CLL RAL             /ACCUMULATE LSB(PRODUCT)
        DCA PROD
        SZL                 /* OVERFLOW TEST
        JMP ERROR           /*
        TAD ML2             /GET NEXT BIT OF MULTIPLIER
        RAL
        DCA ML2
        SZL                 /IF SET, ADD MULTIPLICAND INTO
                            /THE PARTIAL PRODUCT
        TAD I ML            /MULTIPLICAND
        CLL                 /* OVERFLOW TEST
        TAD PROD            /PARTIAL PRODUCT
        ISZ COUNT           /LOOP FOR 12 BITS
        JMP ML1
        ISZ ML
        SNL                 /* OVERFLOW TEST
        JMP I ML            /RETURN
        CLA                 /*
        JMP ERROR           /*

/VARIABLES

ML2,    0                   /MULTIPLIER
PROD,   0                   /PARTIAL PRODUCT
COUNT,  0
KM14,   -14
```

B.2 UNSIGNED FRACTIONAL MULTIPLICATION SUBROUTINE

```
/CALCULATES A*B/10000, WHERE A AND B ARE UNSIGNED INTEGERS.

/CALLING SEQUENCE:
/       JMS ML
/       A
/       B
/CONTROL RESUMES HERE WITH AC=A*B/10000.
/A AND B ARE PRESERVED.

/NOTE: THERE ARE NO ERROR CONDITIONS.

ML,     0
        CLA                     /* MAY BE REMOVED IF AC=0 ON ENTRY
        TAD I ML                /GET MULTIPLIER
        ISZ ML
        DCA ML2
        TAD KM14
        DCA COUNT
ML1,    DCA PROD
        TAD ML2                 /GET NEXT BIT OF MULTIPLIER
        CLL RAR
        DCA ML2
        SZL                     /IF SET, ADD MULTIPLICAND INTO
        TAD I ML
        CLL
        TAD PROD                /THE PARTIAL PRODUCT
        RAR                     /KEEP MSB (PROD) ONLY
        ISZ COUNT               /LOOP FOR 12 BITS
        JMP ML1
        ISZ ML
        JMP I ML                /RETURN

/VARIABLES

ML2,    0                       /MULTIPLIER
PROD,   0                       /PARTIAL PRODUCT
COUNT,  0
KM14,   -14
```

B.3 UNSIGNED INTEGER DIVISION SUBROUTINE

```
/CALCULATES A/B,WHERE A AND B ARE UNSIGNED INTEGERS SUCH THAT
/B IS NOT E.0 .  IF B=0 THEN A JMP IS MADE TO LOC 'ERROR'.
/QUOTIENT IS RETURNED IN AC, REMAINDER IS AVAILABLE IN
/LOCATION 'REMAIN'.

/CALLING SEQUENCE:
/       JMS DV
/   .   A
/       B
/CONTROL RESUMES HERE (I.E., AT THE THIRD LOCATION AFTER THE
/JMS INSTRUCTION)WITH AC=QUOTIENT(A/B),REMAIN=REMAINDER(A/B).
/A AND B ARE PRESERVED.  LINK=0 ON RETURN.

/NOTE: STARRED INSTRUCTIONS MAY BE REMOVED IF THEIR EFFECTS
/ARE NOT NEEDED.

DV,     0
        CLA                     /* IGNORE AC
        TAD I DV                /GET ARGS
        ISZ DV
        DCA DA                  /DIVIDEND
        TAD I DV
        ISZ DV
        SNA                     /* B = 0 TEST
        JMP ERROR               /*
        CLL CIA                 /SUBTRACTION WILL BE DONE BY ADDING
        DCA DB                  /-DIVISOR
        DCA REMAIN              /CLEAR MSB(DIVIDEND)
        CLL CLA CMA RAL /SET INITIAL VALUE OF QUOTIENT TO 7776.  AS QUOTIENT
                                /IS SHIFTED LEFT, THE 0 BIT WILL SHIFT LEFT,
                                /SERVING AS A FLAG TO STOP THE DIVISION LOOP
                                /AFTER ALL 12 BITS ARE COMPUTED.
DV1,                            /MAIN LOOP: COMPUTE NEXT BIT OF PARTIAL
        DCA QUOT                /QUOTIENT INITIALIZE OR STORE PARTIAL
        TAD DA                  /QUOTIENT SHIFT REMAINDER LEFT TO NEXT BIT
        CLL RAL
        DCA DA
        TAD REMAIN              /ONE BIT INTO MSB(DIVIDEND)
        RAL
        DCA REMAIN
        TAD REMAIN              /TRIAL SUBTRACTION
        TAD DB                  /SUBTRACT DIVISOR FROM DIVIDEND
        SZL                     /STORE BACK REMAINDER ONLY IF .GE. 0
        DCA REMAIN
        CLA                     /LINK=1 IFF SUBTRACT SUCCEEDED
        TAD QUOT
        RAL                     /ROTATE LINK INTO PARTIAL QUOTIENT
        SZL                     /DO LOOP ONCE FOR EACH BIT
        JMP DV1
        JMP I DV                /RETURN

/VARIABLES

DA,     0                       /LSB(DIVIDEND)
REMAIN, 0                       /MSB(DIVIDEND), REMAINDER
DB,     0                       /-DIVISOR
QUOT,   0                       /PARTIAL QUOTIENT
```

B.4 UNSIGNED FRACTIONAL DIVISION SUBROUTINE

```
/CALCULATES 10000*A/B, WHERE A AND B ARE UNSIGNED INTEGERS SUCH THAT
/B .NE. 0 AND A<B.  IF B=0 OR A .GE. B THEN A JMP IS MADE TO LOCATION "ERROR".

/CALLING SEQUENCE:
/       JMS DV
/       A
/       B
/CONTROL RESUMES HERE (I.E., AT THE THIRD LOCATION AFTER THE
/JMS INSTRUCTION) WITH AC=QUOTIENT(10000*A/B).
/A AND B ARE PRESERVED.  LINK=0 ON RETURN.

/NOTE: STARRED INSTRUCTIONS MAY BE REMOVED IF THEIR EFFECTS
/ARE NOT NEEDED.

DV,     0
        CLA                 /* IGNORE AC
        TAD I DV            /GET ARGS
        ISZ DV
        DCA DA              /DIVIDEND
        TAD I DV
        ISZ DV
        SNA                 /* B = 0 TEST
        JMP ERROR           /*
        CLL CIA             /SUBTRACTION WILL BE DONE BY ADDING
        DCA DB              /-DIVISOR
        TAD DA              /* OVERFLOW TEST
        TAD DB              /*
        SZL CLA             /*
        JMP ERROR           /*
        CLL CLA CMA RAL /SET INITIAL VALUE OF QUOTIENT TO 7776.  AS QUOTIENT
                            /IS SHIFTED LEFT, THE 0 BIT WILL SHIFT LEFT,
                            /SERVING AS A FLAG TO STOP THE DIVISION LOOP
                            /AFTER ALL 12 BITS ARE COMPUTED.
DV1,                        /MAIN LOOP: COMPUTE NEXT BIT OF PARTIAL
        DCA QUOT            /QUOTIENT INITIALIZE OR STORE PARTIAL
        TAD DA              /QUOTIENT SHIFT REMAINDER LEFT TO NEXT BIT
        CLL RAL
        DCA DA
        TAD DA              /TRIAL SUBTRACTION
        TAD DB              /SUBTRACT DIVISOR FROM DIVIDEND
        SZL                 /STORE BACK REMAINDER ONLY IF .GE. 0
        DCA DA
        CLA                 /LINK=1 IFF SUBTRACT SUCCEEDED
        TAD QUOT
        RAL                 /ROTATE LINK INTO PARTIAL QUOTIENT
        SZL                 /DO LOOP ONCE FOR EACH BIT
        JMP DV1
        JMP I DV            /RETURN

/VARIABLES

DA,     0                   /DIVIDEND (REMAINDER)
DB,     0                   /-DIVISOR
QUOT,   0                   /PARTIAL QUOTIENT
```

APPENDIX C
USER SERVICE ROUTINE

The User Service Routine, or USR, is a collection of subroutines that perform the operations of opening and closing files, loading device handlers, program chaining, and calling the Command Decoder. The USR provides these functions not only for the system itself, but for any programs running under the OS/78 system. A summary of USR functions is given in Table C-1.

USR is only used by PAL8 programs. Users who code in BASIC or FORTRAN IV can skip this appendix and Appendix F, "Using Device Handlers".

The USR resides on the system diskette and, when called, is swapped into memory. Typically, a series of calls is involved in any I/O operation and it is inefficient to repeat the swap every time. Therefore, the calling program has the facility to lock USR in memory in the first 2000 (octal) words of field 1, through the use of USRIN function. To perform any operations on files, it is necessary to have the relevant device handlers available.

This is accomplished with the FETCH function which loads a device handler into memory. Since the handlers reside in the calling program, space must be reserved for them. Special characteristics of OS/78 device handlers and their use are described in Appendix F.

An attempt to call the USR with a code greater than 13 (octal) will cause a Monitor Error 4 message to be printed and the program to be aborted.

C.1 CALLING THE USR

Performing any USR function is done by issuing a JMS instruction followed by the proper arguments. Calls to the USR take a standardized calling sequence. This standard call should be studied before progressing to the operation of the various USR functions.

C.1.1 Standard USR Call

In the remainder of this chapter, the following calling sequence is referenced:

TAD VAL	The contents of the AC is applicable in some cases only.
CDF N	Where N is the value of the current program field multiplied by 10 (octal).
CIF 10	The instruction field must be set to 1.
JMS I (USR)	Where USR is either 7700 or 0200, (see Section C.1.2).
FUNCTION	This word contains an integer from 1 to 13 (octal) indicating which USR operation is to be performed as described in the USR Functions Summary Table (Table C-1).
ARG(1), ARG(2), ARG(3)	The number and meaning of these argument words varies with the particular USR function to be performed.
error return	When applicable, this is the return address for any errors.
normal return	The operation was successful. The AC is cleared and the data field is set to current field.

Table C-1. Summary of USR Functions

Function Code	Name	Operation
1	FETCH	Loads a device handler into memory. Return the entry address of the handler.
2	LOOKUP	Search the file directory on any device to locate a specified permanent file.
3	ENTER	Creates and opens for output a tentative file on a specified device.
4	CLOSE	Closes the currently open tentative file on the specified device, making it a permanent file. Also, any previous permanent file with the same file name and extension is deleted.
5	DECODE	Calls the Command Decoder. The function of the Command Decoder is described in Appendix D.
6	CHAIN	Loads a specified memory image file from the system device and starts it.
7	ERROR	Prints an error message of the form USER ERROR n AT LOCATION xxxxx.
10	USRIN	Loads the USR into memory. Subsequent calls to the USR are by an effective JMS to location 10200.
11	USROUT	Dismisses the USR from core and restores the previous contents of locations 10000 to 11777.
12	INQUIRE	Ascertains whether a given device exists and, if so, whether its handler is in memory.
13	RESET	Resets system tables to their initial cleared state.
14 – 17		Not currently used, these request numbers are reserved for future use.

This calling sequence can change from function to function. For example, some functions take no value in the AC and others have fewer or greater numbers of arguments. However, this format is generally followed.

The value of the data field preceding the JMS to the USR is exceedingly important. The data field MUST be set to the current field, and the instruction field MUST be set to 1. Note that a CDF is not explicitly required if the data field is already correct. When a doubt exists as to the data field setting, an explicit CDF should be executed.

There are three other restrictions which apply to all USR calls, as follows:

1. The USR can never be called from any address between 10000 and 11777. Attempting to do so results in the:

   MONITOR ERROR 4 AT xxxxx (ILLEGAL USR CALL)

message and termination of program execution. The value of xxxxx is the address of the calling sequence (in all such MONITOR ERROR messages).
2. Several USR calls take address pointers as arguments. These pointers always refer to data in the same memory field as the call.
3. When calling the USR from field 1, these address pointers must never refer to data that lies in the area 10000 to 11777.

C.1.2 Direct and Indirect Calling Sequence

A user program can call the USR in two ways. First, by performing a JMS to location 17700. In this case, locations 10000 to 11777 are saved on a special reserved area on the system device, and the USR is then loaded into 10000 to 11777. When the USR operation is completed, locations 10000 to 11777 are restored to their previous values.

NOTE

By setting bit 11 of the Job Status Word to a 1, you can avoid this saving and restoring of memory for programs that do not use locations 0000 to 1777 in field 1.

Alternatively, a program can keep the USR permanently resident in memory at locations 10000 to 11777 by using the USRIN function (see section C.2.8). Once the USR has been brought into memory, a USR call is made by performing a JMS to location 10200. This is the most efficient way of calling the USR. When USR operations have been completed, the program restores locations 10000 to 11777 to their original state by executing the USROUT function, if necessary (see Section C.2.9).

C.2 USR FUNCTIONS

C.2.1 FETCH Device Handler Function Code = 1

Device handlers must be loaded into memory to make them available to the USR and user program for I/O operations on that device. Before performing a LOOKUP, ENTER, or CLOSE function on any device, the handler for that device must be loaded by FETCH.

The FETCH function takes two distinct forms:

1. Load a device handler corresponding to a given device name.
2. Load a device handler corresponding to a given device number.

First, the following is an example of loading a handler by name from memory field 0:

```
CLA             /AC MUST BE CLEAR
CDF 0           /DF = CURRENT FIELD
CIF 10          /IF = 1
JMS I (USR
1               /FUNCTION CODE = 1
DEVICE RXA1     /GENERATES TWO WORDS: ARG(1)
                /AND ARG(2)
6001            /ARG(3)
JMP ERR         /ERROR RETURN
.               /NORMAL RETURN
.
.
```

ARG(1) and ARG(2) contain the device name in standard format. If the normal return is taken, ARG(2) is changed to the device number corresponding to the device loaded. ARG(3) contains the following information:

Bits 0 to 4 contain the page numer into which the handler is to be loaded (handlers are always loaded and used in field 0).

Bit 11 is 0 if the user program can only accept a 1-page handler for this FETCH operation.

Bit 11 is 1 if there is room for a 2-page handler.

Notice that in the example above, the handler for RXA1 is to be loaded into locations 6000 to 6177. If necessary, a two page handler could be loaded; the second page would be placed in locations 6200 to 6377. After a normal return, ARG(3) is changed to contain the entry point of the handler.

A different set of arguments is used to fetch a device handler by number. The following is an example of this form:

```
TAD VAL         /AC IS NOT ZERO
CDF 0           /DF = CURRENT FIELD
CIF 10          /IF = 1
JMS I (USR)
1               /FUNCTION CODE = 1
6001            /ARG(1)
JMP ERR         /ERROR RETURN
.               /NORMAL RETURN
.
.
```

On entry to the USR, the AC contains the device number in bits 8 to 11 (bits 0 to 7 are ignored). The format for ARG(1) is the same as that for ARG(3) in the previous example. Following a normal return ARG(1) is replaced with the entry point of the handler.

The conditions that can cause an error return to occur in both cases are as follows:

1. There is no device corresponding to the given device name or device number, or
2. An attempt was made to load a two page handler into one page. If this is an attempt to load the handler by name, the contents of ARG(2) have been changed already to the internal device number.

In addition, one of the following Monitor errors can be printed, followed by a return to the Keyboard Monitor:

Error Message	Meaning
MONITOR ERROR 4 AT xxxxx (ILLEGAL USR CALL)	Results if bits 8 to 11 of the AC are zero (and bits 0 to 7 are non-zero).
MONITOR ERROR 5 AT xxxxx (I/O ERROR ON SYS)	Results if a read error occurs while loading the device handler.

The FETCH function checks to see if the handler is in memory, and if it is not, then the handler and all co-resident handlers are loaded. While the FETCH operation is essentially a simple one, the following points should be noted:

1. Device handlers are always loaded into memory field 0.
2. The entry point that is returned may not be on the page desired. This would happen if the handler were already resident.

3. Never attempt to load a handler into page 37 or into page 0. Never load a two page handler into page 36 since this will corrupt the OS/78 resident monitor.

For more information on using device handlers, see Appendix F.

NOTE

Two or more device handlers are "co-resident" when they are both included in the same one or two memory pages, for example, RXA0 and RXA1.

C.2.2 LOOKUP Permanent File Function Code = 2

This request locates a permanent file entry on a given device, if one exists. An example of a typical LOOKUP would be:

```
                TAD VAL             /LOAD DEVICE NUMBER
                CDF 0               /DF = CURRENT FIELD
                CIF 10              /IF = 1
                JMS I (USR)
                2                   /FUNCTION CODE = 2
                NAME                /ARG(1), POINTS TO FILE NAME
                0                   /ARG(2)
                JMP ERR             /ERROR RETURN
                ...                 /NORMAL RETURN
NAME,           FILENAME PROG.PA
```

This request looks up a permanent file with the name PROG.PA. The device number on which the lookup is to be performed must be in AC bits 8 to 11 when the call to USR is made. ARG(1) must contain a pointer to the file name. Note that the file name block must be in the same memory field as the call, and that it cannot be in locations 10000 to 11777. The device handler must have been previously loaded into memory. If the normal return is taken, ARG(1) is changed to the starting block of the file and ARG(2) is changed to the file length in blocks as a negative number. If the device specified is a readable, non-file structured device (for example, the terminal), then ARG(1) and ARG(2) are both set to zero.

If the error return is taken, ARG(1) and ARG(2) are unchanged. The following conditions cause an error return:

1. The device specified is a write-only device.
2. The file specified was not found.

In addition, specifying illegal arguments can cause one of the following monitor errors, followed by a return to the Keyboard Monitor:

Error Message	Meaning
MONITOR ERROR 2 AT xxxxx (DIRECTORY I/O ERROR)	Results if an I/O error occurred while reading the device directory.
MONITOR ERROR 3 AT xxxxx (DEVICE HANDLER NOT IN CORE)	Results if the device handler for the specified device is not in memory.
MONITOR ERROR 4 AT xxxxx (ILLEGAL USR CALL)	Results if bits 8 to 11 of the AC are zero.

The LOOKUP function is the standard method of opening a permanent file for input.

C.2.3 ENTER Output (Tentative)File Function Code = 3

The ENTER function is used to create a tentative file entry to be used for output. An example of a typical ENTER function is as follows:

```
            TAD VAL         /AC IS NOT ZERO
            CDF 0           /DF = CURRENT FIELD
            CIF 10          /IF = 1
            JMS I (USR)
            3               /FUNCTION CODE = 3
            NAME            /ARG(1) POINTS TO FILE NAME
            0               /ARG(2)
            JMP ERROR       /ERROR RETURN
            .               /NORMAL RETURN
            .
            .
NAME,       FILENAME COS.DA
```

Bits 8 to 11 of the AC contain the device number of the selected device; the device handler for this device must be loaded into memory before performing an ENTER function. If bits 0 to 7 of the AC are non-zero, this value is considered to be a declaration of the maximum length of the file. The ENTER function searches the file directory for the smallest empty file that contains at least the declared number of blocks. If bits 0 to 7 of the AC are zero, the ENTER function locates the largest available empty file.

On the normal return, the contents of ARG(1) are replaced with the starting block of the file. The 2's complement of the actual length of the created tentative file in blocks (which can be equal to or greater than the requested length) replaces ARG(2).

NOTE

If the selected device is not file structured but permits output operations (for example, a line printer), the ENTER operation always succeeds. In this case, ARG(1) and ARG(2) are both zeroed on return.

If the error return is taken, ARG(1) and ARG(2) are unchanged. The following conditions cause an error return:

1. The device specified by bits 8 to 11 of the AC is a read only device.
2. No empty file exists which satisfies the request length requirement.
3. Another tentative file is already active on this device (only one output file can be active at any given time).
4. The first word of the file name was 0 (an illegal file name).

In addition, one of the following monitor errors can occur, followed by a return to the Keyboard Monitor:

Error Message	Meaning
MONITOR ERROR 2 AT xxxxx (DIRECTORY I/O ERROR)	Result if an I/O error occurred while reading or writing the device directory.
MONITOR ERROR 3 AT xxxxx (DEVICE HANDLER NOT IS MEMORY)	Results if the device handler for the specified device is not in memory.
MONITOR ERROR 4 AT xxxxx (ILLEGAL USR CALL)	Results if AC bits 8 to 11 are zero.

Error Message	Meaning
MONITOR ERROR 5 AT xxxxx (I/O ERROR ON SYS)	Read error on the system device while bringing in the overlay code for the ENTER function.
MONITOR ERROR 6 AT xxxxx (DIRECTORY OVERFLOW)	Results if a directory overflow occurred (no room for tentative file entry in directory).

C.2.4 The CLOSE Function Function Code = 4

The CLOSE function has a dual purpose: first, it is used to close the current active tentative file, making it a permanent file. Second, when a tentative file becomes permanent it is necessary to remove (delete) any permanent file having the same name; this operation is also performed by the CLOSE function. An example of CLOSE usage follows:

```
          TAD VAL           /GET DEVICE NUMBER
          CDF 0             /DF = CURRENT FIELD
          CIF 10            /IF = 1
          JMS I (USR)
          4                 /FUNCTION CODE = 4
          NAME              /ARG(1)
          15                /ARG(2)
          JMP ERR           /ERROR RETURN
          .                 /NORMAL RETURN
          .
NAME,     FILENAME COS.DA
```

The device number is contained in AC bits 8 to 11 when calling the USR. ARG(1) is a pointer to the name of the file to be closed and/or deleted and ARG(2) contains the number of blocks to be used for the new permanent file.

The normal sequence of operations on an output file is:

1. FETCH the device handler for the output device.
2. ENTER the tentative file on the output device, getting the starting block and the maximum number of blocks available for the file.
3. Perform the actual output using calls to the device handler, keeping track of how many blocks are written, and checking to insure that the file does not exceed the available space.
4. CLOSE the tentative file, making it permanent. The CLOSE operation would always use the same file name as the ENTER performed in step 2. The closing file length would have been computed in step 3 and used in the CLOSE call.

After a normal return from CLOSE, the active tentative file is permanent and any permanent file having the specified file name already stored on the device is deleted. If the specified device is a non-file structured device that permits output (the lineprinter, for example) the CLOSE function will always succeed.

NOTE

The user must be careful to specify the same file names to the ENTER and the CLOSE functions. Failure to do so can cause several permanent files with identical names to appear in the directory. If CLOSE is intended only to be used to delete some existing file, then the number of blocks, ARG(2) should be zero.

The following conditions cause the error return to be taken:

1. The device specified by bits 8 to 11 of the AC is a read only device.
2. There is neither an active tentative file to be made into a permanent file, nor a permanent file with the specified name to be deleted.

In addition, one of the following Monitor errors can occur:

Error Message	Meaning
MONITOR ERROR 1 AT xxxxx (CLOSE ERROR)	Results if the length specified by ARG(2) exceeded the allotted space.
MONITOR ERROR 2 AT xxxxx (DIRECTORY I/O ERROR)	Results if an I/O error occurred while reading or writing the device directory.
MONITOR ERROR 3 AT xxxxx (DEVICE HANDLER NOT IS MEMORY)	Results if the device handler for the specified device is not in memory.
MONITOR ERROR 4 AT xxxxx (ILLEGAL USR CALL)	Results if AC bits 8 to 11 are zero.

C.2.5 Call Command Decoder (DECODE) Function Code = 5

The DECODE function causes the USR to load and execute the Command Decoder. The Command Decoder accepts (from the terminal) a list of input and output devices and files, along with various options. The Command Decoder performs a LOOKUP on all input files, sets up necessary tables in the top page of field 1, and returns to the user program.

A typical call to the Command Decoder looks as follows:

```
CDF 0           /DF = CURRENT FIELD
CIF 10          /IF = 1
JMS I (USR
5               /FUNCTION CODE = 5
2001            /ARG(1), ASSUMED INPUT EXTENSION (.PA)
0               /ARG(2), ZERO TO PRESERVE ALL
                /TENTATIVE FILES
.               /NORMAL RETURN
.
.
```

ARG(1) is the assumed input extension; in this example it is ".PA". On return from the Command Decoder, information is stored in tables located in the last page of memory field 1. The DECODE function also resets all system tables as in the RESET function (see RESET function, Section C.2.11); if ARG(2) is 0 all currently active tentative files remain open; if ARG(2) is non-zero all tentative files are deleted and the normal return is to ARG(2) instead of ARG(2)+1.

The DECODE function has no error return (Command Decoder error messages are given in Appendix D). However, the following Monitor error can occur:

Error Message	Meaning
MONITOR ERROR 5 AT xxxxx (I/O ERROR ON SYS)	I/O error occurred while reading or writing on the system device.

C.2.6 CHAIN Function Function Code = 6

The CHAIN function permits a program to run another program with the restriction that the program chained to must be a memory image (.SV) file located on the system device. A typical implementation of the CHAIN function follows:

```
CDF 0           /DF = CURRENT FIELD
CIF 10          /IF = 1
JMS I (USR
6               /FUNCTION CODE = 6
BLOCK           /ARG(1), STARTING BLOCK NUMBER
```

There is no normal or error return from CHAIN. However, the following monitor error can occur:

Error Messages	Meaning
MONITOR ERROR 5 AT xxxxx (I/O ERROR ON SYS)	I/O error occurred while reading or writing on the system device.
CHAIN ERR	If an attempt is made to CHAIN to a file which is not a memory image (.SV) file. Control returns to the Keyboard Monitor.

The CHAIN function loads a memory image file located on the system device beginning at the block number specified as ARG(1) (which is normally determined by performing a LOOKUP on the desired file name). Once loaded, the program is started at an address one greater than the starting address specified by the program's Core Control Block.

CHAIN automatically performs a USROUT function (see Section C.2.9), if necessary, to dismiss the USR from memory, and a RESET to clear all system tables (see Section C.2.11), but CHAIN does not delete tentative files. Normally, it is best to set bit 11 of the Job Status Word before chaining to an OS/78 system program.

The areas of memory altered by the CHAIN function are determined by the contents of the Core Control Block of the memory image file loaded by CHAIN. The Core Control Block for the file is set up by ABSLDR or LOADER programs. It can be modified by performing a SAVE command with specific arguments. Every page of memory in which at least one location was saved is loaded. If the page is one of the "odd numbered" pages (pages 1, 3, etc.; locations 0200 to 0377, 0600 to 0777, etc.), the previous page is always loaded. In addition, CHAIN always alters the contents of locations 07200 to 07577.

NOTE

CHAIN destroys a necessary part of the ODT resident breakpoint routine. Thus an ODT break-point routine. Thus an ODT breakpoint should never be attempted across a CHAIN.

With the above exceptions, programs can pass data back and forth in memory while chaining. For example, FORTRAN programs normally leave the COMMON area in memory field 1 unchanged. This COMMON area can then be accessed by the program activated by the CHAIN.

C.2.7 Signal User ERROR Function Code = 7

The USR can be called to print a user error message for a program. The following is a possible ERROR call:

```
CDF 0           /DF = CURRENT FIELD
CIF 10          /IF = 1
JMS I (USR)
7               /FUNCTION CODE = 7
2               /ARG(1), ERROR NUMBER
```

The ERROR function causes a message of the form:

USER ERROR n AT xxxxx

to be printed. Here n is the error number given as ARG(1); n must be between 0 and 11 (octal), and xxxxx is the address of ARG(1). If ARG(1) in the sample call above was at location 500 in field 0, the message:

USER ERROR 2 AT 00500

would be printed. Following the message, the USR returns control to the Keyboard Monitor, preserving the user program intact.

The error number is arbitrary. Two numbers have currently assigned meanings:

Error Message	Meaning
USER ERROR 0 AT xxxxx	During a RUN, GET, or R command, this error message indicates that an error occurred while loading the memory image.
USER ERROR 1 AT xxxxx	While executing a FORTRAN program, this error indicates that a call was made to a subroutine that was not loaded.

C.2.8 Lock USR In Memory (USRIN) Function Code = 10

When making a number of calls to the USR it is advantageous for a program to avoid reloading the USR each time a USR call is made. The USR can be brought into memory and kept there for subsequent use by the USRIN function. The calling sequence for the USRIN function looks as follows:

```
CDF 0           /DF = CURRENT FIELD
CIF 10          /IF = 1
JMS I (7700)
10              /FUNCTION CODE = 10
.               /NORMAL RETURN
.
.
```

The USRIN function saves the contents of locations 10000 to 11777 in a reserved area on SYS: (provided the calling program loads into this area as indicated by the current Job Status Word), then loads the USR, and finally returns control to the user program.

NOTE

If bit 11 of the current Job Status Word is a one, the USRIN function will not save the contents of locations 10000 through 11777.

C.2.9 Dismiss USR From Memory (USROUT) Function Code = 11

When a program has loaded the USR into memory with the USRIN function and no longer wants or needs the USR in memory, the USROUT function is used to restore the original contents of locations 10000 to 11777. The calling sequence for the USROUT function is as follows:

```
CDF 0             /DF = CURRENT FIELD
CIF 10            /IF = 1
JMS I (200)       /DO NOT JMS TO 17700!!
11                /FUNCTION CODE = 11
.                 /NORMAL RETURN
.
.
```

Subsequent calls to the USR must be made by performing a JMS to location 7700 in field 1.

NOTE

If bit 11 of the current Job Status Word is a 1, the contents of memory are not changed by the USROUT function. In this case USROUT is a redundant operation since memory was not preserved by the USRIN function.

C.2.10 Ascertain Device Information (INQUIRE) Function Code = 12

On some occasions a user may wish to determine what internal device number corresponds to a given device name or whether the device handler for a specified device is in memory, without actually performing a FETCH operation. INQUIRE performs these operations for the user. The function call for INQUIRE closely resembles the FETCH handler call.

INQUIRE, like FETCH, has two distinct forms:

1. Obtain the device number corresponding to a given device name and determine if the handler for that device is in memory (example shown below).
2. Determine if the handler corresponding to a given device number is in memory.

An example of the INQUIRE call is shown below:

```
CLA               /AC MUST BE CLEAR

CDF 0             /DF = CURRENT FIELD
CIF 10            /IF = 1
JMS I (USR)
12                /FUNCTION CODE = 12
DEVICE RXA1       /GENERATES TWO WORDS:
                  /ARG(1) AND ARG(2)
0                 /ARG(3)
JMP ERR           /ERROR RETURN
.                 /NORMAL RETURN
.
.
```

ARG(1) and ARG(2) contain the device name in standard format. When the normal return is taken ARG(2) is changed to the device number corresponding to the given name, and ARG(3) is changed to either the entry point of the device handler if it is already in memory, or zero if the corresponding device handler has not yet been loaded.

A slightly different set of arguments is used to inquire about a device by its device number:

```
TAD VAL         /AC IS NON-ZERO
CDF 0           /DF = CURRENT FIELD
CIF 10          /IF = 1
JMS I (USR)
12              /FUNCTION CODE = 12
0               /ARG(1)
JMP ERR         /ERROR RETURN
.               /NORMAL RETURN
.
.
```

On entry to INQUIRE, AC bits 8 to 11 contain the device number.

NOTE

If AC bits 0 to 7 are non-zero, and bits 8 to 11 are zero (an illegal device number), the

MONITOR ERROR 4 AT xxxxx

message is displayed and program execution is terminated.

On normal return ARG(1) is set to the entry point of the device handler if it is already in memory, or zero if the corresponding device handler has not yet been loaded. The error return in both cases is taken only if there is no device corresponding to the device name or number specified.

C.2.11 RESET System Tables Function Code = 13

There are certain occasions when it is desired to reset the system tables, effectively removing from memory all device handlers except the system handler. An example of the RESET function is shown below:

```
CDF 0           /DF = CURRENT FIELD
CIF 10          /IF = 1
JMS I (USR)
13              /FUNCTION CODE = 13
0               /O PRESERVES TENTATIVE FILES
.               /NORMAL RETURN
.
.
```

RESET removes all device handlers, other than that for the system device, from memory. This should be done anytime a user program modifies any page in which a device handler was loaded.

RESET has the additional function of deleting all currently active tentative files (files that have been entered but not closed). This results in zeroing bits 9 through 11 of every entry in the Device Control Word Table (see Section B.3.5 of *OS/8 Software Support Manual).* If RESET is to be used to delete all active tentative files, then ARG(1) must be non-zero and the normal return is to ARG(1) rather than to ARG(1)+1. For example, the following call would serve this purpose

```
CDF 0           /DF:CURRENT FIELD
CIF 10          /IF = 1
JMS I (USR
13              /FUNCTION CODE = 13
CLA CMA         /NON-ZERO INSTRUCTION
```

The normal return would execute the CLA CMA and all active tentative files on all devices would be deleted. The Keyboard Monitor currently does not reset the Monitor tables. If user programs which do not call the Command Decoder are used, it is wise to do a RESET operation before loading device handlers. The RESET will ensure that the proper handler will be loaded into memory.

APPENDIX D
THE COMMAND DECODER

OS/78 provides a powerful subroutine called the Command Decoder for use by all system and user programs. The Command Decoder is normally called when a program starts running. When called, the Command Decoder displays an asterisk (*) and then accepts a command line from the console terminal that includes a list of I/O devices, file names, and various option specifications. The Command Decoder validates the command line for accuracy, performs a LOOKUP on all input files, and sets up various tables for the calling program (refer to Appendix C, Section C.2.2, for more information on LOOKUP).

D.1 COMMAND DECODER CONVENTIONS

The command line has the following general form in response to the Command Decoder asterisk:

```
*output files < input files/ (options)
```

There can be 0 to 3 output files and 0 to 9 input files specified.

Output File Examples	Meaning
EXPLE.EX	Output to a file named EXPLE.EX on device DSK (the default file storage device).
LPT:	Output to the LPT. This format generally specifies a non-file structured device.
RXA2:EXPLE.EX	Output to a file named EXPLE.EX on device RXA2.
RXA2:EXPLE.EX[99]	Output to a file named EXPLE.EX on device RXA2. A maximum output file size of 99 blocks is specified.
(null)	Not output specified.

An input file specification has one of the following forms:

Input File Format	Meaning
RXA2:INPUT	Input from a file named INPUT.df on device RXA2. "df" is the assumed input file extension specified to the Command Decoder.
RXA2:INPUT.EX	Input from a file named INPUT.EX on device RXA2. In this case .EX overrides the assumed input file extension.
INPUT.EX	Input from a file named INPUT.EX. If there is no previously specified input device, input is from device DSK, the default file storage device; otherwise, the input device is the same as the last specified input device.
TTY:	Input from terminal; no file name is needed for non-file structured devices.
(null)	Repeats input from the previous device specified (must not be first in input list, and must refer to a non-file structured device).

NOTE

Whenever a file extension is left off an input file specification, the Command Decoder first performs a LOOKUP for the given name appending a specified assumed extension. If the LOOKUP fails, a second LOOKUP is made for the file appending a null (zero) extension.

The Command Decoder verifies that the specified device names, file names, and extensions consist only of the characters A through Z and 0 through 9. If not, a syntax error is generated and the command line is considered to be invalid.

There are two kinds of options that can be specified: first, alphanumeric option switches are denoted by a single alphanumeric character preceded by a slash (/) or a string of characters enclosed in parentheses; secondly, a numeric option can be specified as an octal number from 1 to 37777777 preceded by an equal sign (=). These options are passed to the user program and are interpreted differently by each program.

Finally, the Command Decoder permits the command line to be terminated by either the RETURN or ESCape key. This information is also passed to the user program.

D.2 COMMAND DECODER ERROR MESSAGES

If an error in the command line is detected by the Command Decoder, one of the following error messages is displayed. After the error message, the Command Decoder starts a new line, prints an *, and waits for another command line. The erroneous command is ignored.

Error Message	Meaning
ILLEGAL SYNTAX	The command line is formatted incorrectly.
TOO MANY FILES	More than three output files or nine input files were specified. (Or in special mode, more than 1 output file or more than 5 input files.)
device DOES NOT EXIST	The specified device name does not correspond to any permanent device name or any user assigned device name.
name NOT FOUND	The specified input file name was not found on the selected device.

D.3 CALLING THE COMMAND DECODER

The Command Decoder is initiated by the DECODE function of the USR. DECODE causes the contents of locations 0 to 1777 of field 0 to be saved on the system scratch blocks, and Command Decoder to be brought into that area of memory and started. When the command line has been entered and properly interpreted, the Command Decoder exits to the USR, which restores the original contents of 0 to 1777 and returns to the calling program.

NOTE

By setting bit 10 of the Job Status Word to a 1 the user can avoid this saving and restoring of memory for programs that do not occupy locations 0 to 1777.

The DECODE call can reside in the area between 00000 to 01777 and still function correctly. A typical call would appear as follows:

```
CDF 0        /SET DATA FIELD TO CURRENT FIELD
CIF 10       /INSTRUCTION FIELD MUST BE 1
JMS I (USR /USR=7700 IF USR IS NOT IN MEMORY
             /OR USR=0200 IF USRIN WAS PERFORMED
5            /DECODE.FUNCTION = 5
2001         /ARG(1),ASSUMED INPUT EXTENSION
0            /ARG(2),ZERO TO PRESERVE
             /ALL TENTATIVE FILES
.            /NORMAL RETURN
.
.
```

ARG(1) is the assumed input extension in SIXBIT notation. If an input file name is given with no specified extension, the Command Decoder first performs a LOOKUP for a file having the given name with the assumed extension. If the LOOKUP fails, the Command Decoder performs a second LOOKUP for a file having the given name and a null (zero) extension. In this example, the assumed input extension is ".PA".

DECODE performs an automatic RESET operation to remove from memory all device handlers except those equivalent to the system device. As in the RESET function, if ARG(2) is zero all currently active tentative files are preserved. If ARG(2) is non-zero, all tentative files are deleted and DECODE returns to ARG(2) instead of ARG(2)+1.

As the Command Decoder normally handles all of its own errors, there is no error return from the DECODE operation.

D.4 COMMAND DECODER TABLES

The Command Decoder sets up various tables in the top page of field 1 that describe the command line typed to the user program.

D.4.1 Output Files

There is room for three entries in the output file table that begins at location 17600. Each entry is five words long and has the following format:

	0 1 2 3 4 5 6 7	8 9 10 11	
WORD 1	USER SPECIFIED FILE LENGTH	4-BIT DEVICE NUMBER	BITS 0-7 ARE ALWAYS 0

	0 1 2 3 4 5	6 7 8 9 10 11	
WORD 2	FILE NAME CHARACTER 1	FILE NAME CHARACTER 2	OUTPUT FILE NAME 6 CHARACTER
WORD 3	FILE NAME CHARACTER 3	FILE NAME CHARACTER 4	
WORD 4	FILE NAME CHARACTER 5	FILE NAME CHARACTER 6	
WORD 5	FILE EXTENSION CHARACTER 1	FILE EXTENSION CHARACTER 2	FILE EXTENSION 2 CHARACTERS

Bits 0 to 7 of word 1 in each entry contain the file length, if the file length was specified with the square bracket construction in the command line. Otherwise, those bits are zero.

The entry for the first output file is in locations 17600 to 17604, the second is in locations 17605 to 17611, and the third is in locations 17612 to 17616. If word 1 of any entry is zero, the corresponding output file was not specified. A zero in word 2 means that no file name was specified.

Also, if word 5 of any entry is zero no file extension was specified for the corresponding file. It is left to the user program to take the proper action in these cases.

These entries are in a format that is acceptable to the ENTER function.

D.4.2 Input Files

There is room for nine entries in the input file table that begins at location 17617. Each entry is two words long and has the following format:

	0 1 2 3 4 5 6 7	8 9 10 11
WORD 1	MINUS FILE LENGTH	4-BIT DEVICE NUMBER
WORD 2	STARTING BLOCK OF FILE	

Bits 0 to 7 of word 1 contain the file length as a negative number. Thus, 377 (octal) in these bits is a length of one block, 376 (octal) is a length of two blocks, etc. If bits 0 to 7 are zero, the specified file has a length greater than or equal to 256 blocks or a non-file structured device was specified.

NOTE

This restriction to 255 blocks of actual specified size can cause some problems if the program has no way of detecting end-of-file conditions.

If this is liable to be a problem, it is suggested that the user program employ the special mode of the Command Decoder described in Section D.5 and perform its own LOOKUP on the input files to obtain the exact file length.

The two-word input file list entries beginning at odd numbered locations from 17617 to 17637 inclusive. If location 17617 is zero, no input files were indicated in the command line. If less than nine input files were specified, the unused entries in the input file list are zeroed (location 17641 is always set to zero to provide a terminator even when no files are specified).

D.4.3 Command Decoder Option Table

Five words are reserved beginning at location 17642 to store the various options specified in the command line. The format of these five words is as follows:

	0	1	2	3	4	5	6	7	8	9	10	11
17642	HIGH ORDER 11 BITS OF = N OPTIONS											
17643	A	B	C	D	E	F	G	H	I	J	K	L
17644	M	N	O	P	Q	R	S	T	U	V	W	X
17645	Y	Z	0	1	2	3	4	5	6	7	8	9
17646	LOW ORDER 12 BITS OF = N OPTIONS											

Each of the bits in 17643 and 17645 corresponds to one of the possible alphanumeric option switches. The corresponding bit is 1 if the switch was specified, 0 otherwise.

NOTE

If no = n option is specified, the Command Decoder zeroes 17646 and bits 1 to 11 of 17642. Thus, typing = 0 is meaningless since the user program cannot tell that any option was specified.

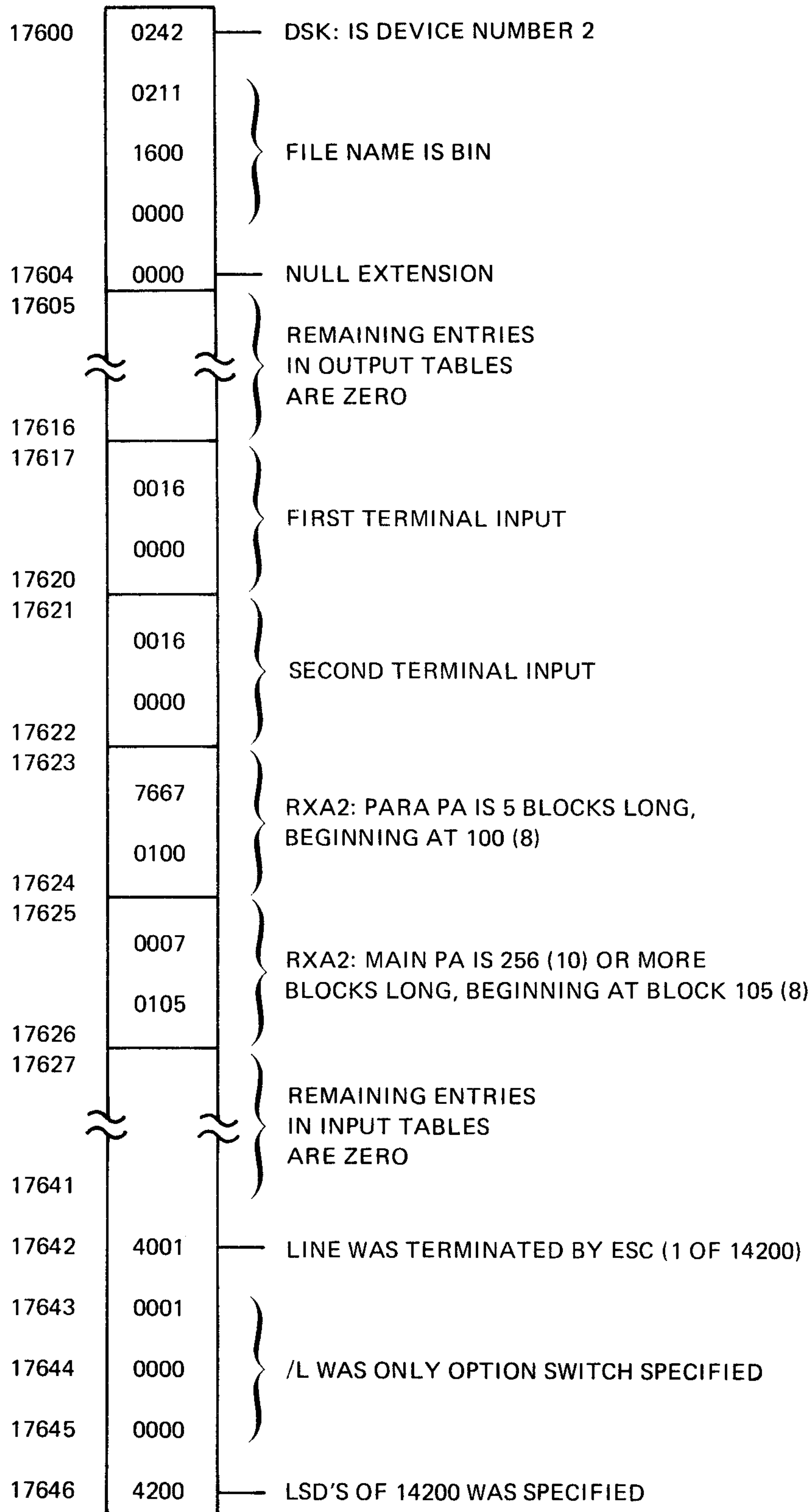
Bit 0 of location 17642 is 0 if the command line was terminated by a carriage return, 1 if it was terminated by an ESCape.

D.4.4 Example

To clarify some of the preceding, consider the interpretation of the following command line:

```
*BIN[10]<TTY:,,RXA2:PARA.PA,MAIN.PA /L=14200
```

If this command line is typed, the Command Decoder would return to the calling program with the following values in the system tables:



NOTE

The entries for terminal (where no input file name is specified) have a starting block number and file size of zero. This is always true of the input table for a non-file structured device, or a file structured device on which no file name is given.

D.5 SPECIAL MODE OF THE COMMAND DECODER

Occasionally the user program does not want the Command Decoder to perform the LOOKUP on input files, leaving this option to the user program itself. Programs such as format conversion routines which access non-standard file structures could use this special format. If the input files were not OS/78 format, a command decoder LOOKUP operation would fail. The capability to handle this case is provided in the OS/78 Command Decoder. This capability is generally referred to as the "special mode" of the Command Decoder.

D.5.1 Calling the Command Decoder Special Mode

The special mode call to the Command Decoder is identical to the standard DECODE call except that the assumed input file extension, specified by ARG(1), is equal to 5200. The value 5200 corresponds to an assumed extension of ".*", which is illegal. Therefore, the special mode of the Command Decoder in no way conflicts with the normal mode.

D.5.2 Operation of the Command Decoder in Special Mode

In special mode the Command Decoder is loaded and inputs a command line as usual. The appearance of the command line is altered by the special mode in these respects:

1. Only one output file can be specified.
2. No more than five input files can be specified, rather than the nine acceptable in normal mode.
3. The characters asterisk (*) and question mark (?) are legal in file names and extensions, both in input files and on output files. It is strongly suggested that these characters be tested by the user program and treated either as special options or as illegal file names. The user program must be careful not to ENTER an output file with an asterisk or question mark in its name as such a file cannot easily be manipulated or deleted by the standard system programs.

The output and option table set up by the Command Decoder is not altered in special mode. Entries in the input table are changed to the following format:

	0 1 2 3 4 5	6 7 8 9 10 11	
WORD 1		4-BIT DEVICE NUMBER	BITS 0-7 ARE ALWAYS 0
WORD 2	FILE NAME CHARACTER 1	FILE NAME CHARACTER 2	INPUT FILE NAME 6 CHARACTER
WORD 3	FILE NAME CHARACTER 3	FILE NAME CHARACTER 4	
WORD 4	FILE NAME CHARACTER 5	FILE NAME CHARACTER 6	
WORD 5	FILE EXTENSION CHARACTER 1	FILE EXTENSION CHARACTER 1	FILE EXTENSION 2 CHARACTERS

The table entry for the first input file is in locations 17605 to 17611; the second in locations 17612 to 17616; the third in location 17617 to 17623; the fourth in locations 17624 to 17630; and the fifth in locations 17631 to 17635. A zero in word 1 terminates the list of input files. If word 2 of an entry is zero, no input file name was specified.

APPENDIX E
OS/78 DEFAULT FILE NAME EXTENSIONS

This appendix lists the default file name extensions used in OS/78.

Extension	Meaning
.BA	BASIC source file (default extension for a BASIC input file).
.BI	Batch input file (input for BATCH).
.BN	Absolute binary file (default extension for ABSLDR and BITMAP input files; also used as default extension for PAL8 binary output file).
.CM	Command file.
.DI	Directory listing file.
.FT	FORTRAN language source file (default extension for FORTRAN input files).
.LD	FORTRAN load module file (default assumed by run-time system, FORTRAN IV loader).
.LS	Assembly listing Output file (default extension for PAL8).
.MP	File containing a loading map (used by the Linking Loader, MAP command).
.PA	PAL8 source file.
.RA	RALF assembly language file (FORTRAN IV).
.RL	Relocatable binary file (default extension for a Linking Loader input file).
.SV	Memory image file (SAVE file); default for the R, RUN, SAVE, and GET commands.
.TM	Temporary file generated by system.

APPENDIX F
USING DEVICE HANDLERS

A device handler is a system subroutine that is used by all parts of the OS/78 system and by all standard system programs to perform I/O transfers. All device handlers are called in the same way and they all perform the same basic operation of reading or writing a specified number of 128 word records beginning at a selected memory address. This appendix contains information relevant to assembly language (PAL8) programming only.

These subroutines effectively mask the unique characteristics of different I/O devices from the calling program; thus, programs that use device handlers properly are effectively "device independent". Changing devices involves merely changing the device handlers used for I/O.

Device handlers have another important feature. They are able to transfer a number of records as a single operation. On a diskette, a single operation could transfer an entire track or more. This capability significantly increases the speed of operation of programs that have large buffer areas.

All device handlers occupy either one or two memory pages, and can run in any page of field 0, except in Page 0 and 37, and in the case of two-page handlers, 36.

NOTE

The word "record" is defined to mean 128 words of data; thus, an OS/78 block consists of two 128-word records.

F.1 CALLING DEVICE HANDLERS

Device handlers are loaded into a user selected area in memory field 0 by the FETCH function (this is possible since SYS: is always resident). FETCH stores in ARG(1) the entry point of the handler loaded. The handler is called by executing a JMS to the specified entry point address. Handler calls have the following format:

```
        CDF N           /WHERE N IS THE VALUE OF THE CURRENT
                        /PROGRAM FIELD TIMES 10 (OCTAL)
        CIF 0           /DEVICE HANDLER ALWAYS IN FIELD 0
        JMS I ENTRY     /"ENTRY CONTAINS THE HANDLER ENTRY POINT
        ARG(1)          /FUNCTION CONTROL WORD
        ARG(2)          /BUFFER ADDRESS
        ARG(3)          /STARTING BLOCK NUMBER
        JMP ERR         /ERROR RETURN
        .               /NORMAL RETURN (I/O TRANSFER COMPLETE)
        .
        .
ENTRY,  0               /ENTRY CONTAINS ARG(3) FROM THE FETCH OPERATION
```

NOTE

The entry point for SYS is always 07607.

As with calls to the USR, it is important that the data field is set to the current program field before the device handler is called. On exit from the device handler, the data field will remain set to current program field. The accumulator need not be zero when calling a handler; it will be zero when the normal return is taken.

ARG(1) is the function control word, and contains the following information:

Bits	Contents
Bit 0	0 for an input operation (read), 1 for an output operation (write).
Bits 1 to 5	The number of 128 word records to be transferred. If bits 1-5 are zero and the device is non-file structured (i.e., TTY, LPT, etc.) the operation is device dependent. If the device is file structured (SYS, diskette, RXA1), a read/write of 40 (octal) pages is performed.
Bits 6 to 8	The memory field in which the transfer is to be performed.
Bits 9 to 11	Unused (device dependent) bits should be left zero.

ARG(2) is the starting location of the transfer buffer.

ARG(3) is the number of the block at which the transfer is to begin. The user program initially determines this value by performing a LOOKUP or ENTER operation. After each transfer the user program should itself add to the current block number (this argument) the actual number of blocks transferred, equal to one-half the number of 128 word records specified, rounded up if the number of records was odd.

There are two kinds of error returns: fatal and non-fatal. When an error return occurs and the contents of the AC are negative, the error is fatal. A fatal error can be caused by a bad data checksum (CRC) or an attempt to write on a read-only device (or vice versa). The meaning can vary from device to device, but in all cases it is serious enough to indicate that the data transferred, if any, is invalid.

When an error return occurs and the contents of the AC are greater than or equal to zero, a non-fatal error has occurred. This error always indicates detection of the end-of-file character (CTRL/Z) on non-file-structured input devices. For example, when a CTRL/Z is typed during input from device TTY, the TTY handler inserts a CTRL/Z code in the buffer and takes the error exit with the AC equal to zero. While all non-file-structured input devices can detect the end-of-file condition, no file structured device can; furthermore, no device handler takes a non-fatal error return when doing output.

The following restrictions apply to the use of device handlers:

1. If bits 1 to 5 of the function control word, ARG(1), are zero, a transfer of 40 (octal) pages or an entire memory field is indicated. Care must be used to ensure that the handler is not overlaid in this call. This only applies to file-structured handlers. This usage is not recommended.
2. The user program must never specify an input into locations 07600 to 07777, 17600 to 17777, or 27600-27777, or the page(s) in which the device handler itself resides. In general, 7600-7777 in every memory field are reserved for use by system software. Those areas should be used with caution.
3. Note that the amount of data transferred is given as a number of 128 word records, exactly one half of an OS/78 block. Attempting to output an odd number of records will change the contents of the last 128 words of the last block written.
4. The specified buffer address does not have to begin at the start of a page. The specified buffer cannot overlap fields, rather the address will "wrap around" memory. For example, a write of 2 pages starting at location 07600 would cause locations 07600-07777 and 00000-00177 of field 0 to be written.
5. If bits 1-5 of the function control word ARG(1) are zero, a device-dependent operation occurs. Users should not expect a 40-page (full field) transfer of data. The CLOSE operation of the USR calls the handler with bits 1-5 and 9-11 of the function control word 0. This condition means 'perform any special close operations desired'. Non-file structured handlers which need no special handling on the conclusion of data transfers should treat this case as a NOP. An example of usage of such special codes would be where the line printer would perform a line feed.

F.2 OS/78 DEVICE HANDLERS

This section describes briefly the operation of standard OS/78 device handlers, including normal operation, any special initialization operations for function control word = 0, terminating conditions, and response to control characters typed at the keyboard.

F.2.1 Line Printer (LPT, LQP)

1. Normal Operation

   These handlers unpack characters from the buffer and print them on the line printer. The character horizontal tab (ASCII 211) causes sufficient spaces to be inserted to position the next character at a "tab stop" (every eighth column, by definition). The character vertical tab (ASCII 213) causes a skip to the next paper position for vertical tabulation if the line printer hardware provides that feature. The character form feed (ASCII 214) causes a skip to the top of the next page. Finally, the handler maintains a record of the current print column and starts a new line after the full width has been printed. This handler functions properly only on ASCII data.

2. Initialization for Function Control Word = 0

   Before printing begins, the line printer handler issues a form feed to space to the top of the next page.

3. Terminating Condition

   On detection of a CTRL/Z character in the buffer, the line printer handler issues a form feed and immediately takes the normal return. Attempting to input from the line printer forces a fatal error to be returned. A fatal error is also returned if the line printer error flag is set. There are no non-fatal errors associated with the line printer handler.

4. Terminal Interaction

   Typing CTRL/C forces a return to the Keyboard Monitor.

F.2.2 File-structured Devices (SYS, DSK, RXA0, RXA1, RXA2, RXA3)

1. Normal Operation

   These handlers transfer data directly between the device and the buffer.

2. Initialization for Function Control Word = 0

   None.

3. Terminating Conditions

   A fatal error is returned whenever the transfer causes one of the error flags in the device status register to be set. For example, a fatal error would result if a CRC error occurred while reading data from a diskette. The device handlers generally try three times to perform the operation before giving up and returning a fatal error. There are no non-fatal errors associated with file-structured devices.

4. Terminal Interaction

   Typing CTRL/C forces a return to the Monitor.

NOTE

The system device handler (SYS) does NOT respond to a typed CTRL/C.

F.2.3 Terminal Handlers (TTY, SLU2, SLU3)

Listed are the features of the terminal handlers:

1. It reads a line at a time. Whenever the user types CR, it enters CR, LF into the buffer; it echoes CR, LF; and then pads the remainder of the buffer with nulls and returns to the calling program. The characters get put into the buffer one character per word. Thus every third character is a null so far as OS/78 is concerned.
2. The DELETE erases the previous character. The character will disappear from the screen.
3. CTRL/U echoes as ^U and erases the current line, allowing the user to retype it. (It also starts a new line.) The buffer pointer is reset to the beginning of the buffer.
4. CTRL/Z echoes as ^Z (followed by CR, LF) and is used to signal end-of-file (end of input). The ^Z enters the buffer and the remainder of the buffer is padded with nulls. The error return is taken with a positive AC (non-fatal error).
5. Nulls are ignored.
6. CTRL/C echoes as ^C and returns control to the keyboard monitor via location 07600.

On output: (either normal output or when echoing input)

1. CTRL/C on keyboard echoes as ^C and returns control to the Monitor.
2. CTRL/O on keyboard stops further echoing. All echoing ceases (through possibly many buffer loads) until either the handler is reloaded into memory or the user types any character on the keyboard. Not operative during input.
3. CTRL/S causes the handler to suspend output to the terminal. No characters are lost and output resumes when a CTRL/Q is typed. CTRL/S and CTRL/Q do not echo. These characters are operative only upon output. On input, they are treated like other input characters. This is useful on high speed CRT displays.

F.2.4 Multiple Input Files

There is a peculiarity associated with reusing Device Handler areas in OS/78. This problem is best illustrated by the following example:

Assume a program has reserved locations 1000-1377 for its input handler and locations 7400-7577 for its output handler. If the program gives a USR FETCH command to load the RXA1 handler as an input device handler, both RX handlers will load into 1000-1377, since they are both co-resident. If another FETCH is issued to load the RXA0 handler as an output device handler, that handler will not be loaded, because it shares space with the RXA1 handler currently in memory. This is fine; however, if the user now switches input devices and FETCHes the terminal handler as an input device handler it will destroy the RXA0 handler and the next attempt to output using the RXA0 handler will produce errors. There are two ways to get around this problem.

1. Always assign the handler which you expect to stay in memory the longest first. Most programs can process more than one input file per program step (for example, an assembly pass is one program step) but only one output file; therefore, they assign the output handler before any of the input handlers. In the above example, the problem would be eliminated if the RXA1 handler were assigned first.
2. Always give a USR RESET call before each FETCH. Obviously, this call should not delete any open output files. This means that the USR will always load the new handler, even if another copy is in memory. The user must FETCH the output handler again before issuing the USR CLOSE call, otherwise the USR will determine that the output handler is not in core and give a MONITOR ERROR 3 message.

APPENDIX G
OS/78 ERROR MESSAGE SUMMARY

Error messages generated by OS/78 programs are listed in alphabetic order and identified by the system program by which they are generated. Also, system halts that occur as a result of errors are listed. This appendix is only a summary. Refer to the appropriate chapters for more detailed information about specific error conditions.

G.1 SYSTEM HALTS

Errors that occur as a result of a major I/O failure on the system device can cause a system halt. These are as follows:

Value of PC	Meaning
00601	A read error occurred while attempting to load ODT.
07461	An error occurred while reading a program into memory during a CHAIN.
07605	An error occurred while attempting to write the Monitor area onto the system scratch blocks.
07702	A user program has performed a JMS to 7700 in field 0. This is a result of trying to call the USR without first performing a CIF 10.
07764	A read error occurred while loading a program.
07772	A read error occurred on the system scratch area while loading a program.
10066	An input error occurred while attempting to restore the USR.
10256	A read error occurred while attempting to load the Monitor.
17676	An error occurred while attempting to read the Monitor from the system device.
17721	An error occurred while saving the USR area.
17727	An error occurred while attempting to read the USR from the system device.
17736	An error occurred while reading the scratch blocks to restore the USR area.

After bootstrapping and retrying the operation which caused the failure, if the error persists, it is the result of a hardware malfunction or a parity error in the system area. Run the appropriate diagnostic program to check the device and rebuild the system.

G.2 ERROR MESSAGES

Message	Program	Explanation
?0	SRCCOM	Insufficient memory – this means that the differences between the files are too large to allow for effective comparison. Use of the /X option may alleviate this problem.
0	Editor	Editor failed in reading a device. Error occurred in device handler; most likely a hardware malfunction.
?1	SRCCOM	Input error on file #1 or less than 2 input files specified.
1	Editor	Editor failed in writing onto a device. Generally a hardware malfunction.
?2	SRCCOM	Input error on file #2.
2	Editor	File close error occurred. The output file could not be closed; the file has been deleted.
2045 REFS	CREF	More than 2044 (decimal) references to one symbol were made.
?3	SRCCOM	Output file too large for output device. It has been deleted.
3	Editor	File open error occurred. This error occurs if the output device is a read-only device or if no output file name is specified for a file-oriented output device.
?4	SRCCOM	Output I/O error.
4	Editor	Device handler error occurred. The Editor could not load the device handler for the specified device. This error should never occur.
?5	SRCCOM	Could not create output file.
AA	F4	More than six subroutine arguments are arrays.
ALREADY EXISTS (filename)	FOTP	An attempt was made to rename an output file with the name of an existing output file.
ARE YOU SURE?	PIP	Occurs when using the SQUISH command. A response of Y will compress the files.
AS	F4	Bad ASSIGN statement.
BAD ARG	FRTS	Illegal argument to library function.
BAD ARGS	Monitor	The arguments to the SAVE command are not consistent and violate restrictions.
BAD DATE	Monitor	The date has not been entered correctly, or incorrect arguments were used, or the date was out of range.

Message	Program	Explanation
BAD DEVICE	CCL	The device specified in a system command is not of the correct form.
BAD DIRECTORY ON DEVICE #n	PIP	PIP is trying to read the directory, but it is not a legal OS/78 directory.
BAD EXTENSION	CCL	Either an extension was specified without a file name or two extensions were specified.
BAD INPUT DIRECTORY	DIRECT FOTP	The directory on the specified input device is not a valid OS/78 directory.
BAD INPUT FILE	Loader	An input file was not a RALF module.
BAD INPUT, FILE #n	ABSLDR	Attempt was made to load a non-binary file as file number n of the input file list; or a non-memory image with /I option.
BAD INPUT, FILE #n	BITMAP	A physical end of file was reached before a logical end of file, or extraneous characters were found in binary file n.
#BAD LINE. JOB ABORTED	BATCH	The BATCH monitor detected a record in the input file that did not have one of the characters dot, slash, dollar sign, or asterisk as the first character of the record. The record is ignored, and BATCH scans the input file for the next $JOB record.
BAD NUMBER	CCL	The =n option was used, where n was not in the correct octal format.
BAD OUTPUT DEVICE	FOTP	This message usually appears when a non-file structured device is specified as the output device for a file-oriented operation.
BAD OUTPUT DEVICE	Loader	The loader image file device was not a directory device, or the symbol map file device was a read-only device. The entire command is ignored.
BAD OUTPUT DIRECTORY	FOTP	The directory on the specified output device is not a valid OS/78 device directory.
BAD RECOLLECTION	CCL	An attempt was made to use a previously remembered argument after the system date had been changed.
BAD SWITCH OPTION	CCL	The character used with a slash (/) to indicate an option is not a legal option.
BATCH.SV NOT FOUND ON SYS:	BATCH	A copy of BATCH.SV must exist on the system device. Control returns to the OS/78 Monitor.
%BATCH SQUISHING SYS:!	BATCH	Batch is running and attempting to squish the system.
BD	F4	Bad dimensions (too big, or syntax) in DIMENSION, COMMON or type declarations.

Message	Program	Explanation
BE	RALF	Illegal equate. The symbol had been defined previously.
	PAL8	PAL8 internal table has overflowed. Fatal error; assembly cannot continue.
BI	RALF	Illegal index register specification.
BO	FRTS	No more file buffer available.
BS	F4	Non-declarative statement used BLOCK DATA program.
BX	RALF	Bad expression. Something in the expression is incorrect, or the expression is not valid in this context.
CANNOT CHANGE MEMORY CAPACITY WHILE RUNNING BATCH	CCL	A MEMORY command was issued while BATCH was running.
?CAN'T-DEVICE DOESN'T EXIST	SET	A nonexistent device was referenced.
?CAN'T-DEVICE IS RESIDENT	SET	No modifications are allowed to the system handler.
?CAN'T-OBSOLETE HANDLER	SET	The handler has an old version number.
?CAN'T-UNKNOWN VERSION OF$THIS HANDLER	SET	This handler's version number is not recognized; possibly a newer version.
CAN'T OPEN OUTPUT FILE	PIP	Message occurs due to one of the following: 1. Output file is on a read-only device. 2. No name has been specified for the output file. 3. Output file has zero free blocks.
CAN'T READ IT	FRTS	I/O error on reading loader image file.
%CAN'T REMEMBER	CCL	The argument specified in a CCL command line is too long to be remembered or an I/O error occurred.
CF	PAL8	CREF.SV not on SYS:
CH	BCOMP	Error in CHAIN statement: attempt to chain from .SV to .BA or vice-versa.
	PAL8	Chain to CREF error – CREF.SV was not found on SYS:.

Message	Program	Explanation
CHAIN ERROR	USR Monitor	Chain error
CI	BRTS	Inquire failure in CHAIN. Device not found.
CL	BRTS	Lookup failure in CHAIN. Filename not found.
	F4	Bad COMPLEX literal.
CLOSE ERROR	USR Monitor	Close error
CLOSE FAILED	CREF	CLOSE on output file failed.
CO	F4	Syntax error in COMMON statement.
COMMAND LINE OVERFLOW	CCL	The command line specified with the @ construction is more than 512 characters in length.
COMPARE ERROR	RXCOPY	A compare error has been detected at the track and sector specified.
CONTRADICTORY SWITCHES	CCL	Either two CCL processor switches were specified in the same command line or the file extension and the processor switch do not agree.
CORE IMAGE ERR	Monitor	Cannot run system program.
CX	BRTS	Chain error
DA	BRTS	Attempt to read past end of data list.
DA	F4	Bad syntax in DATA statement.
DE	BCOMP	Error in DEF statement.
	BRTS	Device driver error. Caused by hardware I/O failure.
	F4	This type of statement illegal as end of DO loop.
	PAL8	Device error. An I/O failure was detected when trying to read or write a device. Fatal error-assembly cannot continue.
DELETES PERFORMED ONLY ON INPUT DEVICE GROUP 1 CAN'T HANDLE MULTIPLE DEVICE DELETES	FOTP	More than one input device was specified in the DELETE command when no output specification (device or filename) was included.
DEV LPT BAD	CREF	The default output device, LPT, cannot be used as it is not available on this system.

Message	Program	Explanation
DEV NOT IMPLE-MENTED	BATCH	BATCH cannot accept input from the specified input device because its handler is not SYS: Control returns to the Command Decoder.
DEVICE DOES NOT HAVE A DIRECTORY	DIRECT	The input device is a non-directory device; for example, TTY.DIRECT can only read directories from file-structured devices.
DEVICE FULL	HELP	Ouptut device storage capacity exhausted.
DEVICE HANDLER NOT IN CORE	USR Monitor	Handler for device specified not in memory.
DEVICE IS NOT RX	RXCOPY	One or more of the devices specified to RXCOPY were not RX diskettes.
DF	PAL8	Device full. Fatal error – assembly cannot continue.
	F4	Bad DEFINE FILE statement.
D.F. TOO BIG	FRTS	Product of number of records times number of blocks per record exceeds number of blocks in file.
DH	F4	Hollerith field error in DATA statement.
DI	BCOMP	Error in DIM statement syntax or string dimension greater than 72, or array dimensioned twice.
DIRECTORY I/O ERROR	USR Monitor	An I/O error occurred while attempting to read or write a directory block.
DIRECTORY OVERFLOW	USR Monitor	Directory overflow occurred.
DIVIDE BY	FRTS	Attempt to divide by zero. The resulting quotient is set to zero and execution continues.
DL	F4	Data list and variable list are not same length.
DN	F4	DO-end missing or incorrectly nested. It is followed by the statement number of the erroneous statement rather than the ISN.
DO	BRTS	No more room for drivers. Too many different devices used in file commands.
	F4	Syntax error in DO or implied DO.
name DOES NOT EXIST	CCL	The device with the name given is not present on the OS/78 system.

Message	Program	Explanation
DP	F4	DO loop parameter not integer or real.
DV	BRTS	Attempt to divide by 0. Result is set to zero (NF).
EF	BRTS	Logical end of file. Usually caused when I/O device runs out of medium.
EG	RALF	The preceding line contains extra code which could not be used by the assembler.
EM	BRTS	Attempt to raise a negative number to a power.
EN	BRTS	Enter error in opening file. Device is read only or there is already one tentative file open on that device or file not found.
ENTER FAILED	CREF	Entering an ouptut file was unsuccessful – possibly output was specified to a read-only device.
EOF ERROR	FRTS	End of file encountered on input.
EQUALS OPTION BAD	DIRECT	The =n option is not in the correct range.
ERROR CLOSING FILE	DIRECT	Output device was read-only.
ERROR IN COMMAND	CCL	A command not entered directly from the console terminal is not a legal CCL command. This error occurs when the argument of a UA, UB, or UC command was not a legal command.
ERROR ON INPUT DEVICE SKIPPING (filename)	FOTP	The file specified is not transferred (due to a hardware I/O error), but any previous or subsequent files are transferred and indicated in the new directory. Any file of the same name may have been deleted from the output device.
ERROR ON OUTUT DEVICE	BITMAP	A hardware I/O error occurred while writing on output device.
ERROR ON OUTPUT DEVICE SKIPPING (filename)	FOTP	The file specified is not transferred, but any previous or subsequent files are transferred and indicated in the new directory.
ERROR READING INPUT DIRECTORY	DIRECT FOTP	A hardware I/O error occurred while reading the directory.
ERROR WRITING FILE	DIRECT	A hardware I/O error occurred while writing the output file.
ERROR WRITING OUTPUT DIRECTORY	FOTP	A hardware I/O error has occurred. The output device has probably been corrupted (directory no longer describes current files).
ES	RALF	External symbol error.
EX	F4	Syntax error in EXTERNAL statement.

Message	Program	Explanation
FB	BRTS	FILE busy. Attempt to use a file already in use.
FC	BRTS	OS/78 error while closing variable file. Device is read-only or file already closed.
FE	BRTS	Fetch error in opening file. Device not found, or device handler too big for available space.
FETCH ERROR	HELP	Cannot initialize device handler.
FI	BRTS	Attempt to close or use unopened file.
FILE ERROR	FRTS	Any of the following: a. A file specified as an existing file was not found. b. A file specified as a non-existing file would not fit on the designated device. c. More than 1 nonexistent file was specified on a single device. d. File specification contained "*" as name or extension.
FILE OVERFLOW	FRTS	Attempt to write outside file boundaries.
FL	RALF	An error has occurred in integer-to-read conversion routines.
FM	BRTS	Attempt to fix minus number. Usually caused by negative subscripts or file numbers.
FN	BCOMP	Error in file number of file name designation.
	BRTS	Illegal file number. Only 0, 1, 2, 3, 4 are legal.
FO	BRTS	Attempt to fix number greater than 4095. Usually caused by negative subscripts of file numbers.
FORMAT ERROR	FRTS	Illegal syntax in FORMAT statement.
FP	BCOMP	Incorrect FOR loop parameters or FOR loop syntax.
FP	RALF	A syntax error was encountered in a real constant.
FR	BCOMB	Error in function arguments or function not defined.
FULL *	Editor	The specified output device has become full. The file is closed; the user must specify a new output file.
GR	BRTS	RETURN without a GOSUB.
GS	BRTS	Too many nested GOSUBS. The limit is 10.
GT	F4	Syntax error in GO TO statement.

Message	Program	Explanation
GV	F4	Assigned or computed GO TO variable must be integer or real.
HO	F4	Hollerith field error.
IA	BRTS	Illegal argument in UDEF function call.
IC	RALF	The symbol or expression in a conditional is improperly used, or left angle bracket is missing. The conditional pseudo-op is ignored.
IC	PAL8	Illegal character. The character is ignored and the assembly continued.
ID	PAL8	Illegal redefinition of a symbol.
IE	F4	A hardware I/O error on reading input file. Control returns to the Monitor.
	PAL8	Illegal equals – an attempt was made to equate a variable to an expression containing an undefined term. The variable remains undefined.
IE	RALF	An entry point has not been defined, or is absolute, or also is defined as a common section, or external.
IF	BCOMP	THEN or GOTO missing from IF statement, or bad relational operator.
	BRTS	Illegal file specification.
	F4	Logical IF statement cannot be used with DO, DATA, INTEGER, etc.
II	PAL8	Illegal indirect – an off-page reference was made; a link could not be generated because the indirect bit was already set.
ILLEGAL *	DIRECT FOTP	An asterisk (*) was included in the output file specification or an illegal * was included in the input file name.
ILLEGAL * OR ?	CCL	An * or ? was used in a CCL command that does not accept the wild card construction.
ILLEGAL ?	DIRECT FOTP	A question mark (?) was included in the output file specification.
ILLEGAL ARG	Monitor	The SAVE command was not expressed correctly; illegal syntax used.
#ILLEGAL INPUT	BATCH	A file specification designated TTY or LPT as an input device when the initial dialogue indicated that an operator is not available. The current job is aborted, and BATCH scans the input file for the next $JOB command record.
ILLEGAL ORIGIN	Loader	A RALF routine tried to store data outside the bounds of its overlay.

Message	Program	Explanation
ILLEGAL SPOOL DEVICE	BATCH	The device specified as a spooling output device must be file-structured. Control returns to the Command Decoder.
ILLEGAL SYNTAX	CCL	The command line was formatted incorrectly.
?ILLEGAL WIDTH	SET	A width that was 0 or too large was specified; for the TTY, a width of 128 or one not a multiple of 8 was specified.
IN	BRTS	Inquire failure in opening file. Device not found.
INPUT DEVICE READ ERROR	RXCOPY	Unable to read input devices.
INPUT ERROR	CREF	A hardware I/O error occurred while reading the file.
	FRTS	Illegal character received as input.
INPUT ERROR READING INDIRECT FILE	CCL	CCL cannot read the file specified with the @ construction due to hardware I/O error.
#INPUT FAILURE	BATCH	Either a hardware problem prevented BATCH from reading the next line of the input file, or BATCH read the last record of the input file without encountering a $END command record.
INSUFFICIENT MEMORY FOR BATCH RUN	BATCH	OS/78 BATCH requires 12K of memory to run. Control returns to the OS/78 Monitor. Type MEMORY 0 and try again.
IO	BCOMP	I/O error.
	BRTS	TTY input buffer overflow. Causes input buffer to be cleared and output another ? to be displayed.
	RALF	I/O error (fatal error).
I/O ERROR	FRTS	Error reading or writing a file, tried to read from an output device, or tried to write on an output device.
I/O ERROR, FILE #n	ABSLDR BITMAP	An I/O error has occurred in input file number n.
IO ERROR IN (file name) --CONTINUING	PIP	An error has occurred during a SQUISH transfer.
I/O ERROR ON SYS:	CCL	An error occurred while doing I/O to the system device. The system must be rebootstrapped.
?I/O ERROR ON SYS:	SET	An I/O error occurred while trying to read or rewrite the handler.
I/O ERROR TRYING TO RECALL	CCL	An I/O error occurred while CCL was trying to remember an argument.

Message	Program	Explanation
IP	PAL8	Illegal pseudo-op – a pseudo-op was used in the wrong context or with incorrect syntax.
IX	RALF	An index register was specified for an instruction which cannot accept one.
IZ	PAL8	Illegal page zero reference – The pseudo-op was found in an instruction which did not refer to page zero. The Z is ignored.
LD	PAL8	The /L or /G options have been specified and ABSLDR.SV is not present on the system.
LG	PAL8	Link Generated – only printed if the /E switch was specified to PAL8.
LI	F4	Argument of logical IF is not of type Logical.
LM	BRTS	Attempt to take log of negative number or 0.
LOADER I/O ERROR	Loader	Fatal error message indicating that an error was detected by OS/78 while trying to perform a USR function.
LS	BCOMP	Missing equal sign in LET statement.
LT	BCOMP	Statement too long (greater than 80 characters).
	F4	Input line too long, too many continuations.
	RALF	The line is longer than 128 characters. The first 127 characters are assembled and listed.
#MANUAL HELP NEEDED	BATCH	BATCH is attempting to operate an I/O device, such as a terminal, that will require operator intervention.
MD	BCOMP	Line number defined more than once. YY equals the line number before line in error.
	RALF	The tag on the line has been previously encountered at another location or has been used in a context requiring an absolute expression.
ME	BCOMP	Missing END statement.
MIXED INPUT	Loader	The L option was specified on a line that contained some file other than a library file. The library file (if any) is accepted. Any other input file specification is ignored.
MK	F4	Misspelled keyword.
ML	F4	Multiply-defined line number.
MM	F4	Mismatched parenthesis.

Message	Program	Explanation
MO	BCOMP F4	Operand expected but not found.
MONITOR ERROR 5 AT xxxx (I/O ERROR ON SYS=)	Monitor	A hardware I/O error occurred while doing I/O to the system device.
MONITOR ERROR 6 AT xxxx (DIRECTORY OVERFLOW)	Monitor USR	A directory overflow has occurred (no room for tentative file entry in directory).
#MONITOR OVERLAYED	BATCH	The Command Decoder attempted to call the BATCH monitor to accept and transmit a file specification, but found that a user program had overlayed part or all of the BATCH monitor. Control returns to the monitor level, and BATCH executes the next Monitor command.
MORE CORE REQUIRED	FRTS	The space required for the program, the I/O device handlers (I/O buffers) and the resident Monitor exceeds the available memory.
MP	BCOMP	Missing parenthesis or error in expression within parentheses.
MT	BCOMP F4	Operand of mixed type or operator does not match operands.
MULT SECT	Loader	Any combination of entry point, COMMON section (with the exception of multiple COMMONs) or program section of the same name causes this error.
NE	RALF	Number error. A number out of range was specified or an 8 or 9 occurred in octal radix.
NF	BCOMP	NEXT statement without corresponding FOR statement.
NM	BCOMP	Line number missing after GOTO, GOSUB, or THEN.
NO!!	Monitor	The user attempted to start (with .ST) a program which is no longer current.
NO CCL!	Monitor	CCL.SV is not present on the system device or an I/O error occurred on reading it.
NO DEFINE FILE	FRTS	Direct access I/O attempted without a DEFINE FILE statement.
NO FILES OF THE FORM xxxx	FOTP	No files of the form (xxxx) specified were found.
NO HELP	HELP	No help information is present on the command given.
NO HELP FILE	HELP	No help file is on the system device.
NO/I	BITMAP	Cannot produce a bitmap of an image file.

Message	Program	Explanation
NO/I!	ABSLDR	Use of /I is prohibited after the first file.
NO INPUT	ABSLDR BITMAP	No input or binary file was found on the designated device.
NO INPUT DEVICE	RXCOPY	The command line lacks the required parameters.
NO OUTPUT DEVICE	RXCOPY	The command line lacks the required parameters.
NO MAIN	Loader	No RALF module contained section #MAIN.
NO NUMERIC SWITCH	FRTS	The referenced FORTRAN I/O unit was not specified to the run-time system.
NO ROOM FOR OUTPUT FILE	DIRECT PIP	Either room on device or room in directory is lacking.
NO ROOM IN (file name) -CONTINUING	PIP	Occurs during the SQUISH command. The output device cannot contain all of the files on the input device.
NO ROOM, SKIPPING (filename)	FOTP	No space is available on the output device to perform the transfer. Predeletion may already have occurred.
NOT A LOADER IMAGE	FRTS	The first input file specified to the run-time system was not a loader image file.
name NOT AVAILABLE	Monitor	The device with the name given is not listed in any system table, or it is not available for use at the moment, or the user tried to obtain input from an output-only device.
NOT ENOUGH MEMORY	CCL	The number specified in a MEMORY command is greater than 3.
name NOT FOUND	CCL Monitor	The file name designated in the command was not found in the device directory.
?NUMBER TOO BIG	SET	The number specified was out of range.
OE	BRTS	Driver error while overlaying. Caused by hardware I/O error on SYS.
OF	BCOMP	Output file I/O error.
	F4	I/O error while writing output file. Control returns to the Monitor.
OP	F4	Illegal operator.
OS78 ENTER ERROR	Loader	Fatal error message indicating that an error was detected by OS/78 while trying to perform a USR function.
OT	F4	Type/operator use illegal (for example, A.AND.B where A and/or B not of type Logical).

Message	Program	Explanation
OUT DEV FULL	CREF	The output device is full (directory devices only).
OUTPUT DEVICE READ ERROR	RXCOPY	RXCOPY cannot read output device.
OUTPUT DEVICE WRITE ERROR	RXCOPY	RXCOPY cannot write output device.
OV	BRTS	Numeric or input overflow.
OVER CORE	Loader	The loader image requires more memory than is available.
OVER IMAG	Loader	Output file overflow in the loader image file.
OVER SYMB	Loader	Symbol table overflow. More than 253 (decimal) symbols in one FORTRAN job.
OVERFLOW	FRTS	Result of a computation exceeds upper bound for that class of variable. The result is set equal to zero and execution continues.
PA	BRTS	Illegal argument in POS function.
PARENS TOO DEEP	FRTS	Parentheses nested too deeply in FORMAT statement.
PD	BCOMP	Pushdown stack overflow. Result of either too complex a statement (or statements), too many nested FOR-NEXT loops, or both.
	F4	Compiler stack overflow; statement too big and/or too many nested loops.
PE	PAL8	Current non-zero page exceeded.
PH	F4	Bad program header line.
	PAL8	A conditional assembly bracket is still in effect at the end of the input stream – this is caused by nonmatching < and > characters in the source.
QL	F4	Nesting error in EQUIVALENCE statement.
QS	BCOMP	String literal too long or does not end in quote.
	F4	Syntax error in EQUIVALENCE statement.
RD	F4	Attempt to redefine the dimensions of an array.
	PAL8	A permanent symbol has been redefined using =. The new and old definitions do not match. The redefinition is allowed.
RE	BRTS	Attempt to read past end of file (NF).

Message	Program	Explanation
READ ERR	HELP	Cannot read input device.
	RALF	Relocatability error. A relocatable expression has been used in a context requiring an absolute expression.
RT	F4	Attempt to redefine the type of a variable.
RW	F4	Syntax error on READ/WRITE statement.
SAVE ERROR	Monitor	An I/O error has occurred while saving the program. The program remains intact in memory.
SC	BRTS	String too long (greater than 80 characters) after concatenating.
SE	PAL8	Symbol table exceeded – too many symbols have been defined for the amount of memory available for the symbol table. Fatal error – assembly cannot continue.
SF	F4	Bad arithmetic statement function.
SL	BRTS	String too long or undefined.
SN	F4	Illegal subroutine name in CALL.
SORRY – NO INTERRUPTIONS	PIP	^C (CTRL/C) was typed during a SQUISH; the transfer continues.
#SPOOL TO FILE BTCHA1	BATCH	Where the "A" may be any character of the alphabet and the "1" may be any decimal digit. This message indicates that BATCH has intercepted a non-file structured output file and routed it to the spool device. This is not, generally, an error condition.
SR	BRTS	Attempt to read string from numeric file.
SS	BCOMP	Subscript or function argument error.
	F4	Error in subscript expression; i.e., wrong number, syntax.
ST	BCOMP	Symbol table overflow due to too many variables, line numbers, or literals. Combine lines using backslash (\) to condense program.
ST	BRTS	String truncation on input. Stores maximum length allowed (NF).
ST	F4	Compiler symbol table full, program too big. Causes an immediate return to the Keyboard Monitor.
	RALF	User symbol table overflow (fatal error).
SU	BRTS	Subscript out of DIM statement range.
SW	BRTS	Attempt to write string into numeric file.

Message	Program	Explanation
SWITCH NOT ALLOWED HERE	CCL	Either a CCL option was specified on the left side of the < or was used when not allowed.
SY	BCOMP	System incomplete. System files such as BASIC.SV, BCOMP.SV, and BRTS.SV missing.
	F4	System error; i.e., PASS2O.SV or PASS2.SV missing, or no room for output file. Causes an immediate return to the Keyboard Monitor.
SYM OVERFLOW	CREF	More than 896 (decimal) symbols and literals were encountered. Try again using /M option.
?SYNTAX ERROR	SET	Incorrect format used in SET command or NO specified when not allowed.
#SYS ERROR	BATCH	A hardware I/O error occurred during BATCH operation.
SYSTEM DEVICE ERROR	FRTS	I/O failure on the system device.
SYSTEM ERR	Monitor	An error occurred while doing I/O to the system device. The system should be rebootstrapped.
SYSTEM ERROR	Loader	Fatal error message indicating that an error was detected by OS/78 while trying to perform a USR function.
SYSTEM ERROR – CLOSING FILE	FOTP	Self-explanatory.
TB	BCOMP	Program too big. Condense or CHAIN.
TD	BCOMP	Too much data in program.
	F4	Bad syntax in type declaration statement.
TOO FEW ARGS	Monitor	An important argument has been omitted from a command.
TOO MANY FILES	CCL	Too many files were included in a CCL command.
TOO MANY HANDLERS	FRTS	Too many I/O device handlers are resident in memory, or files have been defined on too many devices.
TOO MANY RALF FILES	Loader	More than 128 input files were specified.
TS	BCOMP	Too many total characters in the string literals.
UD	BCOMP	Error in UDEF statement.
UF	BCOMP	FOR loop without corresponding NEXT statement.

Message	Program	Explanation
?UNKNOWN ATTRIBUTE FOR DEVICE dev	SET	An illegal attribute was specified for the given device.
UNIT ERROR	FRTS	I/O unit not assigned, or incapable of executing the requested operation.
UO	PAL8	Undefined origin – an undefined symbol has occurred in an origin statement.
US	BCOMP	Undefined statement number.
	RALF PAL8	Undefined symbol in an expression.
USER ERROR	FRTS	Illegal subroutine call, or call to undefined subroutine. Execution continues only if the E option was specified.
USER ERROR 0 AT xxxx	Monitor	An input error was detected while loading the program. xxxx refers to the Monitor location where the error was generated.
UU	BCOMP	Incorrect or missing array designator in USE statement.
VE	F4	Version error. One of the compiler programs is absent from SYS.
VR	BRTS	Attempt to read variable length file.
WE	BRTS	Attempt to write past end of file (NF).
WRITE ERR	HELP	Cannot output to device.
XC	BCOMP	Extra characters after the logical end of line.
ZE	PAL8	Page 0 exceeded – same as PE except with reference to page 0.
ZERO SYS?	PIP	If any attempt is made to zero the system device directory, this message occurs. Responding with Y causes the directory to be zeroed; any other character prevents destruction of the system directory.

GLOSSARY

ABSOLUTE ADDRESS
A number that is permanently assigned as the address of a memory storage location.

ACCESS TIME
The interval between the instant at which a data transfer is requested and the instant at which the data actually starts transferring.

ACCUMULATOR
The register in which the hardware arithmetic operations are performed (abbreviated AC).

ADDRESS
1. A name or number which identifies a location in memory, either within a field (12-bits wide) or within all available memory (15-bits wide with left-most bit 0).
2. The part of an instruction that specifies the location of the operand of that instruction.

ALGORITHM
A prescribed sequence of well-defined rules or processes for the solution of a problem.

ALPHANUMERIC
Pertaining to the character set that contains only letters and numbers.

ARGUMENT
A variable or constant which is given in the call of a subroutine as information to it; the independent variables of a function; the known reference factor necessary to find an item in a table or array (i.e., the index).

ARRAY
An ordered set of data values. An n-dimensional array is a table having n dimensions.

ASCII
American Standard Code for Information Interchange. Established by American Standards Association to standardize a binary code for printing and control characters.

ASSEMBLE
To translate from a source program to a binary program by substituting binary operation codes for mnemonic operation codes and absolute or relocatable addresses for symbolic addresses.

ASSEMBLER
A program which translates assembly language instructions into machine language and assigns memory locations for variables and constants.

ASSEMBLY LANGUAGE
A symbolic language that translates directly into machine language instructions. Usually there is a one-to-one relation between assembly language instructions and machine language instructions.

AUTO-INDEXING
When an absolute memory location in the range of 0010 through 0017 is addressed, indirectly, the content of that location is incremented by one, rewritten into that same location and then used as the effective address of the current instruction.

AUXILIARY STORAGE
Storage that supplements memory such as a diskette.

BASIC
A high-level programming language for arithmetic and string computations. It was developed at Dartmouth College and is exceptionally easy to learn.

BATCH PROCESSING
A method of scheduling programs for execution with no user interaction.

BINARY
Pertaining to the number system with a base or radix of two. In this system numbers are represented by strings of 1's and 0's.

BINARY CODE
A code that makes use of exactly two distinct values: 0 and 1.

BIT
Contraction of "Binary digit", a bit is the smallest unit of information in the binary system of notation.

BIT MAP
A method of keeping track of used and unused entities by assigning one bit in a table to each entity.

BLOCK
A set of consecutive machine words, characters or digits handled as a unit, particularly with reference to input and output; an OS/78 block is 400 octal contiguous words (two memory pages).

BOOTSTRAP
A program of instructions that are executed when the START pushbutton is pressed and whose purpose is to load and start of OS/78 Monitor, usually starts a complex system of programs.

BREAKPOINT
A location in a program at which that program's execution may be suspended, so that partial results can be examined via ODT.

BUFFER
An area that is usually used for temporary storage. Buffers are often used to hold data being passed between processes or devices which operate at different speeds or different times.

BUG
A mistake in the design or implementation of a program resulting in erroneous results.

BYTE
A group of binary digits usually operated upon as a unit, especially one of six bits.

CALL
To transfer control to a specified routine; to invoke a system command.

CALLING SEQUENCE
A specified arrangement of instructions and data necessary to pass parameters and control to a given subroutine.

CCL (CONCISE COMMAND LANGUAGE)
A system program that simplifies the entry of OS/78 commands by calling the selected system program and decoding its command string.

CHAINING
A program technique which involves dividing a program into sections with each section terminated by a call to the next section.

CHARACTER
A single letter, number or symbol (printing or non-printing) used to represent information.

CLEAR
To erase or reset the contents of a memory location or hardware register.

CLOCK
A hardware device that generates periodic program interrupts when enabled.

COMPILE
To translate a source program written in a high-level language such as BASIC or FORTRAN into a binary-coded program. In addition to translating the source language, appropriate subroutines may be selected from a subroutine library, and output in binary code along with the main program.

COMPILER
A program which compiles entire high-level language source programs into binary coded programs.

CONCATENATION
The adjacent joining of two strings of characters to produce a longer string.

CODE
To write instructions for a computer using symbols meaningful to the computer or to an assembler, compiler, or other language processor.

COMMAND
A user order to a computer system, using the keyboard or from a Batch input file.

COMMAND DECODER
A part of OS/78 that interprets file and option specification strings typed by the user.

CONDITIONAL ASSEMBLY
The processes of translating specific sections of an assembly language program into machine code only if certain conditions have been met during the assembly process.

CONFIGURATION
A particular selection of computer, peripherals, and interfacing equipment that are to function together.

CONSTANT
Numeric data used but not changed by a program.

CONTROL
Memory address at which the next instruction will be executed. When "control is passed" to a location, the instruction contained in that location will be the next one to be executed.

CONVERSATIONAL PROGRAM
A program which interacts dynamically with on-line users, that is, an interactive program.

COUNTER
A variable or memory location used to control the number of iterations of a program loop.

CPU
Central Processing Unit. The portion of a computer that executes most instructions.

CRASH
Fail Totally. When a system crashes it will not function at all and must be restarted.

CRC
Cycle Redundancy Check. An error detection technology used for detecting incorrectly read bits in the RX78 diskette drive and control.

CREATE
To open, write, and close a file for the first time.

CROSS REFERENCE PROGRAM (CREF)
A program that generates a sequence-numbered listing file and a table that contains line numbers of all referenced user-defined symbols and literals and their usage.

CYCLE TIME
A basic unit of time in a computer, usually equal to the memory read time plus memory write time. Computer instructions usually execute in multiples of the cycle time.

DATA
A general term used to denote any or all computer-represented facts, numbers, letters and symbols.

DEBUG
To detect, locate, and correct mistakes in a program.

DEFAULT
A parameter that is assumed by the system when none is explicitly specified.

DELIMITER
A character that separates and organizes elements of data, particularly the symbols of a programming language.

DEVICE
A peripheral hardware I/O unit and/or a removable date volume mounted on it.

DEVICE CODES
Numbers assigned to each device in the system, used in the computer instructions for those devices.

DEVICE DRIVERS
See Device Handlers.

DEVICE HANDLERS
Routines that perform I/O for specific devices in a standard format. These routines also handle error recovery and provide device independence.

DEVICE INDEPENDENCE
: The ability of a computer system to divert the input or output of an executing program from one device to another, either automatically if the specified device is out of order, or by a keyboard command to the Monitor.

DIAGNOSTIC
: Pertaining to the detection and isolation of hardware malfunctions.

DIGITAL
: Representation of information by discrete units.

DIRECT ACCESS
: Same as Random Access.

DIRECT ADDRESS
: A number that specifies the location of an instruction operand.

DIRECTORY
: A reserved storage area on a mass storage device that describes the layout of the data on that device in terms of file names, length, location, and creation date.

DISKETTE
: A removable data volume, consisting of a thin disk coated with a magnetic data-storage material. Also called a floppy disk.

DUMMY ARGUMENTS
: Symbolic names used only as placeholders for other actual arguments that will be uniformly substituted for them at a later time. Used within programming language function definitions to represent the independent (supplied) variables.

ECHO
: The displaying by the terminal of characters typed on the keyboard.

EDITOR
: A program which interacts with the programmer or typist to enter new programs into the computer and edit them as well as modify existing programs.

EFFECTIVE ADDRESS
: The address actually used to fetch an instruction operand, that is, the specified address modified by indexing or indirect addressing rules.

ENTRY POINT
: A point in a subroutine to which control is transferred when the subroutine is called.

ERROR MESSAGE
: A message from computer system to programmer that reports a hardware malfunction or an incorrect format or operation detected by software.

EXECUTE
: To cause the computer to carry out an instruction; to run a program on the computer.

FIELD

1. A division of memory containing 4096 decimal (numbered 0-7777 octal) storage cells (locations).
2. An element of a format specification, particularly of a record.

FILE

A contiguous block of characters or computer words, particularly where stored on a mass storage device and entered in the device's directory.

FILE NAME

A name of one to six alphanumeric characters chosen by the user to identify a file.

FILE NAME EXTENSION

Two alphanumeric characters chosen by the programmer or provided by OS/78 to describe the format of information in the file.

FILE-STRUCTURED DEVICE

A device on which files may be stored; also contains a file directory.

FIXED POINT

A format in which one or more computer words (two in the case of OS/8 FORTRAN integers), represent a number with a fixed binary point.

FLAG

A variable or register used to record the status of a program or device. In the latter case it is sometimes called a device flag.

FLOATING POINT

A format in which one or more computer words (three in the case of OS/8 FORTRAN Real and OS/8 BASIC numbers) represent a number with a variable binary point, particularly when there is an exponent part and a mantissa part.

FORMAT

1. A description of a set of valid sequences of language or data elements; syntax.
2. A FORTRAN statement which specifies the arrangement of characters to be used to represent a piece of data.

FORTRAN

A high-level language developed for the scientific community, it stands for FORmula TRANslation.

GARBAGE

Undefined or random data or instructions.

HARD COPY

Computer output in the form of printing on paper and generally in readable form such as listings or other documents.

HEAD

A component that reads, records, or erases data on a storage device such as a diskette.

HIGH LEVEL LANGUAGE

A language in which single statements typically result in more than one machine language instruction, for example, BASIC, FORTRAN.

INDIRECT ADDRESS
An address in a computer instruction which indicates a location in memory where the address of the referenced operand is to be found.

INHIBIT
To prevent, suppress or disallow.

INITIALIZATION CODE
Code which sets counters, switches, and addresses to zero or other starting values at the beginning of or at prescribed points in a computer routine.

INPUT BUFFER
A section of memory used for storage of input data.

INPUT
The process of transferring data to memory from a mass storage device or from other peripheral devices into the AC.

INSTRUCTION
One unit of machine language, usually corresponding to one line of assembly language, which tells the computer what elementary operations to do next.

I/O
Input-output. Refers to transfers of data between memory and peripheral devices.

ITERATION
Repetition of a group (loop) of instructions.

INTERACTIVE
Referring to a mode of using a computer system in which the computer and the user communicate via a computer terminal.

INTERRUPT
A hardware facility that executes an effective JMS to location 0 of field 0 upon any of a particular set of external (peripheral) conditions. The processor state is saved so that the interrupted program can be continued following any desired interrupt-level processing.

INTERRUPT DRIVEN
Pertaining to software that uses the interrupt facility of the computer to handle I/O and respond to user requests.

JOB
A unit of code which solves a problem; a program or sequence of programs.

JUMP
A departure from the consecutive sequence of executing instructions. Control is passed to some location in memory which is specified by the jump instruction.

K
In the computer field, two to the tenth power, which is 1024 in decimal notation. Hence, a 4K memory has 4096 words.

KEYBOARD
: On a typing device, the array of buttons which causes character codes for letters, numbers, and symbols to be generated when pressed.

LABEL
: One or more characters used to identify a source language statement on line or label.

LATENCY
: On rotating storage devices, the delay between the instant the device is notified that a transfer is coming and the instant the device is ready to perform the transfer.

LEAST SIGNIFICANT DIGIT
: The rightmost digit.

LIBRARY
: A collection of standard routines which can be incorporated into other programs.

LINE
: A string of characters terminated with a line feed, vertical tab, or form feed character (and usually also a carriage return). The terminator belongs to the line that it terminates.

LINE NUMBER

1. In source languages such as BASIC and FORTRAN, a number which begins a line for purposes of identification. A numeric label.
2. In editors and listings, the number of the line beginning at the first line of the program and counting each line.
3. An automatically-generated indirect address for an off-page reference (PAL8).

LINK

1. A one-bit register that is complemented when overflow occurs in the accumulator.
2. An address pointer to the next element of a list or next block of a file.

LINKAGE
: In programming, code that connects two separately coded routines and passes values and/or control between them, particularly when the routines occupy separate fields.

LIST
: To output out a listing on the terminal or line printer. The listing of a source program is a sequential copy of statements or instructions in the program. Also, a table of data in a program.

LITERAL
: A constant that defines itself by requesting the assembler to reserve storage for it, even though no storage location is explicitly specified in the source program.

LOAD
: To move data into a hardware register or into memory.

LOADER
: A program which takes information in binary format and copies it into memory. An absolute loader loads binary information that has been prepared for the absolute addresses of memory. A relocatable or linking loader loads binary information specified in relative addresses by assigning an absolute address to every relative address.

LOCATION
A numbered or named cell in memory where a word of data or an instruction or address may be stored.

LOOP
A sequence of instructions that is executed repeatedly until a terminating condition occurs. Also, used as a verb meaning to execute this sequence of instructions while waiting for the ending condition.

MACHINE LANGUAGE
The language, particular to each kind of computer, that those computers understand. It is a binary code which contains an operation code to tell the computer what to do and address to tell the computer what data to perform the operation on.

MAP
A diagram representing memory locations and outlining which locations are used by which programs.

MASK
A combination of bits that is used to clear or accept selected portions of any word, character or register while retaining or ignoring other parts. Also, to clear these selected locations with a mask.

MASK STORAGE
A device such as a diskette that stores large amounts of data readily accessible to the central processing unit.

MATRIX
A rectangular array of elements. A two-dimensional table can be considered a matrix.

MEMORY
The main storage in a computer from which instructions must be fetched and executed. Consists of a sequence of consecutively-numbered cells storing one word each.

MEMORY IMAGE FILE
An executable binary code program file created by the system command SAVE from the current contents of memory.

MEMORY MAP
A diagram or table showing specific memory requirements of one or more programs.

MEMORY REFERENCE INSTRUCTION
A computer instruction that accesses the computer memory during its execution, as opposed to a register instruction which only accesses registers in the CPU and I/O instructions which are commands to peripheral devices.

MNEMONIC
A symbolic representation of an operation code or address (for example, X for unknown variable, JMP for jump instruction).

MODE
A state or method of system or program operation.

MODULE
A routine that handles a particular function.

MONITOR
The collection of routines which schedules resources, I/O, system programs, and user programs, and obeys keyboard commands.

MONITOR SYSTEM
: Editors, assemblers, compilers, loaders, interpreters, data management programs and other utility programs all automated for the user by a monitor.

MOST SIGNIFICANT DIGIT
: The leftmost digit.

NESTING
: The inclusion of one program language construction inside another.

NON-DIRECTORY DEVICE
: A device such as a terminal or a line printer that cannot store or retrieve files.

NO-OP
: Contraction of No Operation. An instruction which specifically instructs the computer to delay for one instruction, and then to get the next instruction.

OBJECT CODE
: The result after assembling or compiling source code.

OCTAL
: The number system with a base, or radix, of eight.

ODT
: Octal Debugging Technique, an interactive program for finding and correcting bugs in programs in which the user communicates in octal notation.

OFF-LINE
: Pertaining to equipment, devices or events which are not under direct control of the computer.

ONE'S COMPLEMENT
: A number formed by setting each bit in an input number to the other bit: 1's become 0's and 0's become 1's.

OP-CODE
: The part of a machine language instruction that identifies the operation that the CPU will be required to perform.

OPERAND
: The data to be used when an instruction is executed.

OPERATING SYSTEM
: See MONITOR SYSTEM

OPERATOR
: That symbol or code which indicates an action or operation to be performed (e.g., + or TAD).

ORIGIN
: The absolute address of the beginning of a section of code.

OUTPUT
: The process of transferring data from memory to a mass storage device or to a listing device such as a line printer.

OVERFLOW
: A condition that occurs when a mathematical operation yields a result whose magnitude is larger than the program or system is capable of handling.

PACK
: To conserve storage requirements by combining data.

PAGE
: A 128 (decimal) word section of memory, beginning at an address which is a multiple of 200 (octal). There are 40 (octal) pages in a field, numbered 0-37 (octal).

PAL
: Programming Assembly Language, the name of the assembly language for the DECstation 78.

PARITY BIT
: A bit that indicates whether the total number of binary one digits in a piece of data is even or odd.

PARITY CHECK
: A check that tests whether the sum of all the bits in an entity is odd or even.

PASS
: One complete reading of a set of input data. An assembler usually requires two passes over a source program in order to translate it into binary code.

PATCH
: To modify a routine in an expedient way, usually by modifying the memory image rather than reassembling it.

PERIPHERAL
: In a data processing system, any device distinct from the CPU, which provides the computer system with outside communication.

POINTER
: A location containing the address of another word in memory.

PRINTOUT
: A loose term referring to almost anything printed by a computer peripheral device; any computer-generated hard copy.

PROGRAM
: A unit of instructions and routines necessary to solve a problem.

PROGRAM COUNTER
: A register in the CPU that holds the address of the next instruction to be executed.

PROGRAMMABLE
: Can be controlled by instructions in a program.

PROMPTING CHARACTER
: A character that is displayed on the console terminal and cues the user to perform some action.

PSEUDO-OP
: Contraction of "Pseudo Operation". An assembly language directive which does not directly translate into machine code but gives directions to the assembler on how to assemble the code that follows.

RADIX
: The base of a number system, the number of digit symbols required by a number system. The decimal number system is radix 10.

RANDOM ACCESS
: Pertaining to a storage device where data or blocks of data can be read in no particular order (for example, diskette).

READ
: To transfer information from a peripheral device into memory or a register in the CPU.

RECORD
: A collection of related items of data treated as a unit, such as a line of source code, or a person's name, address, and telephone number.

REDUNDANCY
: In any data, that portion of the total characters or bits that can be eliminated without any loss of information.

REGISTER
: A device usually made of semiconductor components that is capable of storing a specified amount of data, frequently one word.

RELATIVE ADDRESS
: The number that specifies the difference between the actual address and the base address.

RELOCATE
: To move a routine from one portion of storage to another and to adjust the necessary address references so that the routine can be executed in the new location.

RESTART
: To resume execution of a program.

RETURN
: To pass control back to a calling program at a point following the call when a subroutine has completed its execution.

1. The set of instructions at the end of a subroutine that permits control to return to the proper point in the main program.
2. The point in the main program to which control is returned.
3. The name of a key on the terminal.

RING-BUFFER
: A storage area for data accessed on a first-in, first-out basis whose area is reused circularly.

ROUTINE
: A set of instructions arranged in proper sequence to cause the computer to perform a desired task.

RUN
1. A single continuous execution of a program.
2. To perform that execution.

RUN TIME
: The time during which a program is executed.

SCRATCH PAD MEMORY
Any memory or registers used for temporary storage of partial results.

SECTOR
The smallest unit of actual storage on a diskette. Contains 64 (decimal) words.

SEGMENT
To divide information into segments or to store portions of information program on a mass storage device to be brought into memory as needed.

SERIAL ACCESS
Pertaining to the sequential or consecutive transmission of data to or from memory or a peripheral device.

SERIAL TRANSMISSION
A method of information transfer in which the bits composing the character are sent sequentially on a single path.

SERVICE ROUTINE
A program used for general support of the user; I/O routines, diagnostics, and other utility routines.

SHIFT
A movement of bits to the left or right, usually performed in the accumulator.

SIMULATE
To represent the function of a device, system or program with another device, system or program.

SOFTWARE
The executable instructions used in a computer system.

SOURCE LANGUAGE
Any programming language used by the programmer to write a program before it is translated into machine code.

SOURCE PROGRAM
The computer program written in the source language.

SPOOLING
The technique by which output to slow devices is placed into temporary files on mass storage devices to await transmission. This allows more efficient use of the system since programs using low speed devices can run to completion quickly and make room for others.

STATEMENT
An expression or instruction in a source language.

STORAGE
A general term for any device capable of retaining information.

STORE
To enter data into a storage device, especially memory.

STRING
A sequence of characters.

SUBROUTINE
A section of code, usually performing one task, that may be called from various points of a main program.

SUBSCRIPT
A value used to specify a particular item in an array.

SWAPPING
The movement by the monitor of user programs between memory while they are running and a buffer area on a mass storage device when something else is running in that place in memory.

SYMBOLIC ADDRESS
Alphanumeric characters used to represent a storage location in the context of a particular program. It must be translated to an absolute address by the assembler.

SYMBOL TABLE
A memory storage area or a listing that contains all defined symbols and the binary value associated with each one. Mnemonic operators, labels, and user defined symbols are all placed in the symbol table. (Mnemonic operators stay in the table permanently.)

SYNCHRONOUS
Pertaining to circuits or events where all changes occur simultaneously or in definite timed intervals.

SYSTEM
A combination of software and hardware which performs specific processing operations.

SYSTEM DEVICE
A peripheral mass storage device on which the system software resides. Its handler (SYS) is resident in the last page of field 0.

SYSTEM HEAD
The system area reserved on a file-structured device for the OS/78 Keyboard Monitor. It resides on the system device and contains, in addition to the Monitor, programs such as ODT and Command Decoder.

SYSTEM SOFTWARE
DEC-supplied programs which come in the basic software packages. These include editors, assemblers, compilers, loaders, etc.

TABLE
A collection of data sorted for ease of reference, generally a two-dimensional array.

TEMPORARY STORAGE
Storage locations or registers reserved for intermediate results.

TERMINAL
A peripheral device in a system through which data can enter or leave the computer, especially in a display and keyboard.

TEXT
A message or program expressed in characters.

TRACK
The set of all sectors of a diskette accessible at each head position. A diskette contains 77 (decimal) tracks, each containing 26 (decimal) sectors.

TRUNCATION

The reduction of precision by ignoring one or more of the least significant digits without rounding off.

TWO'S COMPLEMENT

A number used to represent the negative of a given value in many computers. This number is formed from the given binary value by changing all 1's to 0's and all 0's to 1's, and then adding 1.

UTILITIES

Programs to perform general useful functions.

VARIABLE

A piece of data whose value changes during the execution, assembly, or compilation of a program.

WORD

A 12-bit unit of data which may be stored in one addressable location in memory or on a peripheral device.

INDEX

ABS function,
  BASIC, 6-43
  FORTRAN IV, 7-16
Absolute value function, BASIC, 6-43
A conversion (FORTRAN IV), 7-47
Addition,
  BASIC, 6-10
  PAL8, 5-15
Addresses, PAL8, 5-6
ALOG function, FORTRAN IV, 7-17
Alphanumeric information, BASIC, 6-12, 6-16
AND, Boolean (PAL8), 5-16
AND group skip instructions, PAL8, 5-23
.AND. (logical operator), FORTRAN IV, 7-32
Angle bracket (<), usage,
  command format, 2-5
  PAL8, 5-20
Arctangent function,
  BASIC, 6-45
  FORTRAN IV, 7-18
Arithmetic expressions, FORTRAN IV, 7-30
Arithmetic functions, BASIC, 6-43
Arithmetic operations,
  BASIC, 6-10
  FORTRAN IV, 7-30
  PAL8, 5-15
Arithmetic statements,
  BASIC, 7-10
  FORTRAN IV, 7-30
Arrays,
  BASIC, 6-31
  BASIC string, 6-32
  FORTRAN IV, 7-30
Array specifications, FORTRAN IV, 7-54
ASCII
  character set, A-1
  conversion, BASIC, 6-45
  source files, 4-1
Assembler, PAL8, 5-1
Assembly errors, PAL8, 5-32
Assembly instructions, PAL8, 5-5
Assembly language function, BASIC, 6-1
Assembly termination, PAL8, 5-32
ASSIGN command, 2-4, 3-2
Assignment statements,
  BASIC, 6-20
  FORTRAN IV, 7-34
ASSIGN statement, FORTRAN IV, 7-36

Asterisk (*) usage, 2-11
  command decoder, C-1
  FORTRAN IV, 7-8, 7-10
  PAL8, 5-6
@ construction, 2-11
ATAN function, FORTRAN IV, 7-18
ATN function, BASIC, 6-45
Autoindexing, PAL8, 5-25

BACKSPACE statement, FORTRAN IV, 7-54
Backup copy, 2-3
BASIC,
  arithmetic, 6-10
  arithmetic functions, 6-43
  commands, 6-53
  editor commands, 6-2
  error messages, 6-57
  files, 6-15, 6-36
  function summary, 6-55
  instructions, 6-1
  overview, 6-1
  running, 6-2, 6-7
  statements, 6-15
  statement summary, 6-53
  strings, 6-12
  subroutines, 6-26
  system components, 6-1
BATCH,
  calling BATCH, 8-1
  demonstration program, 2-3
  error messages, 8-6
  input file, 8-1
  monitor commands, 8-2
  output file, 8-1
  restrictions, 8-7
  run-time options, 8-2
  spooling, 3-55, 8-55
BLOCK DATA statement, FORTRAN IV, 7-62
Boolean AND, PAL8, 5-16
Boolean inclusive OR, PAL8, 5-16
Bracket ([) PAL8, 5-19
Breakpoints, ODT, 9-2
BYE command, BASIC, 6-6

Calling,
  BASIC, 6-2
  BATCH, 8-1
  Editor, 4-2

INDEX (Cont.)

Calling (Cont.),
Keyboard monitor, 2-1
ODT, 9-1
PAL8, 5-1
Calling relationships, FORLIB, 7-16
Calling sequence, FORTRAN IV loader routines, 7-8
CALL statement, FORTRAN IV, 7-62
CALL subroutine, FORTRAN IV, 7-62
Carriage control characters (FORTRAN IV), 7-48
CCL options, 2-8
Chaining, FORTRAN IV, 7-2
CHAIN statement, BASIC, 6-36
Character deletion,
BASIC, 6-7
Editor, 4-3
keyboard monitor, 2-1
Characters,
ASCII, A-1
BASIC format control, 6-14
Editor special, 4-3
ODT special, 9-2
PAL8, 5-9, 5-10
PAL8 special, 5-18
Character search, Editor, 4-11
Character string search, Editor, 4-12
CHR$ function, BASIC, 6-47
Circumflex, ODT, 9-4
CLOCK subroutine, FORTRAN IV, 7-19
CLOSE # statement, BASIC, 6-41
Codes,
ASCII character, A-1
Command Decoder, D-1
calling, D-2
error messages, D-2
example, D-5
input string, D-1
I/O specification options, D-1
special mode, D-7
tables, D-3
Command mode, Editor, 4-2
Commands,
BASIC editing and control, 6-2
ODT, 9-2
OS/78 Monitor,
ASSIGN, 3-2
BASIC, 3-3
COMPARE, 3-4
COMPILE, 3-7
COPY, 3-8
Commands (Cont.),
OS/78 Monitor (Cont.),
CREATE, 3-14
CREF, 3-15
DATE, 3-17
DEASSIGN, 3-18
DELETE, 3-19
DIRECT, 3-21
DUPLICATE, 3-24
EDIT, 3-26
EXECUTE, 3-28
GET, 3-29
HELP, 3-30
LIST, 3-32
LOAD, 3-34
MAP, 3-36
MEMORY, 3-39
ODT, 3-40
PAL, 3-41
R, 3-42
RENAME, 3-43
RUN, 3-44
SAVE, 3-45
SET, 3-48
START, 3-53
SQUISH, 3-54
SUBMIT, 3-55
TERMINATE, 3-57
TYPE, 3-58
UA, UB, UC, 3-60
ZERO, 3-61
special keyboard commands, Editor, 4-2
Command string format, BATCH, 8-1
Command summary, OS/78, 2-15
Editor, 4-6
Comments,
BASIC, 6-30
FORTRAN IV, 7-28
PAL8, 5-10
COMMON statement, FORTRAN IV, 7-55
COMPARE command, 3-4
Compilation of FORTRAN IV source file, 7-5
COMPILE command, 3-7
Compiler, FORTRAN IV, 7-5
error messages, 7-7
run-time options, 7-6
Computed GOTO, FORTRAN IV, 7-37
Conditional assembly pseudo-operators, PAL8, 5-29
Conditional delimiters, PAL8, 5-21

INDEX (Cont.)

Console terminal as I/O device, FORTRAN IV, 7-10
Constants,
  BASIC string, 6-46
  FORTRAN IV, 7-28
Continuation lines, FORTRAN IV, 7-27
CONTINUE statement, FORTRAN IV, 7-41
Control statements,
  BASIC, 6-24
  FORTRAN IV, 7-36
Conversion,
  BASIC string, 6-49
  FORTRAN H Hollerith, 7-47
COPY command, 3-8
Core control block, 2-14
Corrections, monitor commands, 2-6
COS function,
  BASIC, 6-45
  FORTRAN IV, 7-19
Cosine function, BASIC, 6-45
CREATE command, 3-14
CREF command, 3-15
Cross-Reference Program (CREF), 3-15
  error messages, 3-16
  options, 3-15
  output, 5-7
CTRL keys, 2-2
  BASIC, 6-7
  Editor, 4-2
  FORTRAN IV, 7-14
Current location counter, PAL8, 5-12
Current location, ODT, 9-2

DAT$ function, BASIC, 6-47
Data,
  statement, BASIC, 6-18
  statement, FORTRAN IV, 7-58
  transmission statements, FORTRAN IV, 7-42
DATE command, 3-17
Dates, file creation, 3-17
DEASSIGN command, 3-18
Debugging function, BASIC, 6-52
Decimal format, BASIC, 6-8
DECstation 78 minicomputer system, 1-1
DEFINE FILE statement, FORTRAN IV, 7-50
DEF statement, BASIC, 6-34
Default File Name Extensions, E-1
Defaults, 2-12
DELETE command, 3-19
DELETE key, 2-1
  BASIC, 6-7
  Editor, 4-3
Deletion of characters, 2-1
Delimiters,
  BASIC, 6-29
  PAL8 conditional, 5-21
Demonstration program, 2-3
Device control statements, FORTRAN IV, 7-54
Device entry points, F-1
Device handler assignment, FORTRAN IV, 7-11
Device handlers, F-1
Device names,
  assignment of, 2-3, 3-2
  deassignment of, 2-4, 3-18
  keyboard monitor, 2-3
DEVICE pseudo-op, 5-30
Device specifications, FORTRAN IV, 7-11
Dimensioning strings, BASIC, 6-31
DIMENSION statement, FORTRAN IV, 7-54
DIM statement, BASIC, 6-31
Direct assignment statements, PAL8, 5-14
DIRECT command, 3-21
Diskette,
  loading, 2-1
  system device, 2-4
Division,
  BASIC, 6-10
  PAL8, 5-15
  Subroutine, B-1
DO statement, FORTRAN IV, 7-39
Dollar sign ($),
  BATCH usage, 8-2, 8-5
  Editor, 4-12
  ESCape echo, 4-5
  PAL8 usage, 5-21
DOT (.) character,
  Editor (period), 4-3
  Monitor response, 2-1
  PAL8, 5-19
Double quote (") character, PAL8, 5-19
Dummy arguments, FORTRAN IV, 7-59
DUPLICATE command, 3-24

EDIT command, 3-26
Editor, 4-1
  commands, 4-5
  error messages, 4-18
  key commands, 4-2

Editor (Cont.),
options, 4-2
search modes, 4-11
special characters, 4-3
summary of commands, 4-19
EJECT pseudo-op, PAL8, 5-28
END FILE statement, FORTRAN IV, 7-54
End of file, PAL8, 5-30
END statement,
BASIC, 6-28
FORTRAN IV, 7-42
.EQ. (relational operator), FORTRAN IV, 7-32
Equal sign (=)
arithmetic statement, 5-14, 6-12
input/output options, 2-7, 7-11
EQUIVALENCE statement, FORTRAN IV, 7-56
.EQV. (logical operator), FORTRAN IV, 7-32
Error codes, PAL8, 5-32
Error messages,
BASIC, 6-57
BATCH, 8-6
Editor, 4-17
FORTRAN IV, 7-7, 7-10, 7-15
ODT, 9-7
PAL8, 5-32
Error message summary, OS/78, G-1
Errors in typing, BASIC, 6-7
ESCape key, 2-2
Command Decoder, D-2
Editor, 4-12
FORTRAN IV,
PAL8, 5-6
EXECUTE command, 3-28
Exiting BASIC, 6-6
EXP function,
BASIC, 6-43
FORTRAN IV, 7-20
Exponentiation, BASIC, 6-11
Expressions, FORTRAN IV, 7-30
Extensions, file names, 2-4

.FALSE., FORTRAN IV, 7-34
Fields, FORTRAN IV,
logical, 7-47
numeric, 7-45
FILENAME pseudo-op, 5-30
File names, 2-4
FILE # statement, BASIC, 6-36
File pages, Editor, 1-78
File protection, COPY, 3-10
Files, BASIC, 6-15
File specifications, FORTRAN IV, 7-12
File specifications, FRTS, 7-10
File-structured devices, 2-13
Files, unit, 2-13
FIXMRI pseudo-op, PAL8, 5-28
FIXTAB pseudo-op, PAL8, 5-28
FORLIB.RL (FORTRAN IV library of functions and subroutines), 7-16 to 7-22
Format control characters, BASIC, 6-14
Formats,
assembly listing, PAL8, 5-10
Monitor commands, 2-5
FORMAT statement, FORTRAN IV, 7-42
Form feed, PAL8, 5-10
FOR statement, BASIC, 6-21
FORTRAN IV,
compiler, 7-5, 7-23
library, 7-16
loader, 7-8, 7-24
overview, 8-1
RALF postprocessor, 7-5
running programs, 7-22
run-time system (FRTS), 7-10
source language, 7-27
source programs, 7-1
subprograms, 7-24
FRTS (FORTRAN IV run-time system), 7-10
error messages, 7-15
option specifications, 7-14
Functions,
BASIC, 6-42 to 6-52
FORTRAN IV, 7-16 to 7-22
FUNCTION statements, FORTRAN IV, 7-60

.GE. (relational operator), FORTRAN IV, 7-32
GET command, 3-29
G format conversions, FORTRAN IV, 7-46
GOSUB subroutine, BASIC, 6-26
GOTO statement,
BASIC, 6-24
FORTRAN IV, 7-36
.GT. (relational operator), FORTRAN IV, 7-32

Handlers – see Device handlers
H conversion (FORTRAN IV), 7-47
HELP command, 3-30
HELP files, 2-12
Hollerith, FORTRAN IV constants, 7-29

INDEX (Cont.)

IABS, FORTRAN IV, 7-21
IF statement, FORTRAN IV, 7-38
IFDEF pseudo-op (PAL8), 5-29
IF END # statement, BASIC, 6-42
IFIX, FORTRAN IV, 7-21
IFNDEF pseudo-op, PAL8, 5-29
IFNZRO pseudo-op, PAL8, 5-29
IF-THEN statement, BASIC, 6-25
IFZERO pseudo-op, PAL8, 5-29
Image file, FRTS, 7-10
Index, FORTRAN IV, 8-84
Indirect addressing, PAL8, 5-22
Indirect commands, 2-11
Indirect references, ODT, 9-4
Input files,
  BATCH, 8-1
  Editor, 4-1
Input/output specifications,
  commands, 2-5
  wildcard, 2-10
Input/output statements,
  BASIC, 6-18
  FORTRAN IV, 7-50
Input/output transfer microinstructions, PAL8, 5-25
INPUT statement, BASIC, 6-28
INPUT # statement, BASIC, 6-39
Input string, Command Decoder, D-1
Inputting string data, BASIC, 6-29
Instructions, PAL8, 5-38
Integer constants, FORTRAN IV, 7-28
Integer format, BASIC, 6-9
INTEGER function, BASIC, 6-43
Integer variables, FORTRAN IV, 7-30
Interbuffer character string search, Editor, 4-14
Internal statement number (ISN), FORTRAN IV, 7-5
Intrabuffer character string search, Editor, 4-12
I/O – see Input/Output

Job status word, 3-45

Keyboard, 2-1
Keyboard Monitor, 2-1

Labels, PAL8, 5-10
LEN function (string length), BASIC, 6-48
.LE. (relational operator), FORTRAN IV, 7-32
LET statement, BASIC, 6-20
Libraries, FORTRAN IV, 7-16
Line deletion, keyboard monitor, 2-1
LINE FEED key, 2-2
Link generation and storage, PAL8, 5-31
LIST and LISTNH commands, BASIC, 6-4
LIST command, 3-32
Listing files,
  FORTRAN IV, 7-8
  PAL8, 3-41
Listing suppression, PAL8, 5-27
Lists, BASIC, 6-10
Literals, PAL8, 5-19
LOAD command, 3-34
Loader, FORTRAN IV, 7-8
  error messages, 7-10
  image file, 7-8
  run-time options, 7-9
  symbol map output file, 7-8
Loading,
  binary files, 3-34
  FORTRAN IV,
  PAL8, 5-6
  relocatable programs, 3-35
Location counter, resetting (PAL8), 5-26
Logarithm function, BASIC, 6-45
Logical constants, FORTRAN IV, 7-29
Logical expressions, FORTRAN IV, 7-32
Logical fields, FORTRAN IV, 7-47
Logical operators, FORTRAN IV, 7-32
Loops in BASIC program, 6-21
  nesting, 6-22
.LT. (relational operator), FORTRAN IV, 7-32

MAP command, 3-36
  example, 5-7
  output, 3-36
MEMORY command, 3-39
Memory image files, 2-5, 3-45
Memory reference instructions, PAL8, 5-21
Memory reservation, PAL8, 5-27
Microinstructions, PAL8, 5-22
Monitor,
  calling the monitor, 2-1
  commands, 2-4
  file names and extensions, 2-4
  permanent device names, 2-3
  typing conventions, 2-2
Monitor commands, BATCH, 8-2
Multiple file specifications,
  FORTRAN IV,
  PAL8, 5-6

INDEX (Cont.)

Multiplication,
  BASIC, 6-10
  PAL8, 5-15
  Subroutine, B-1
Multistatement lines, PAL8, 5-11

NAME command, BASIC, 6-6
Names,
  devices, 2-3
  files, 2-3
Natural logarithm function,
  BASIC, 6-45
  FORTRAN IV, 7-17
.NE. (relational operator), FORTRAN IV, 7-32
Nested DO loops, FORTRAN IV, 7-40
Nested literals, PAL8, 5-20
Nested loop commands, BASIC, 6-21
Nested pseudo-ops, PAL8, 5-29
Nested subroutines, BASIC, 6-27
NEW command, BASIC, 6-2
NEXT statement, BASIC, 6-24
.NOT. (logical operator), FORTRAN IV, 7-32
Null extension, FORTRAN IV, 7-10
Numbers, BASIC, 6-8
  printing format, 6-16
Number sign (#), Editor, 4-1
Numbers, PAL8, 5-11
Numeric fields, FORTRAN IV, 7-45

Octal constants, FORTRAN IV, 7-29
Octal Debugging Technique (ODT), 9-1
  commands, 9-2
  errors, 9-4
  special characters, 9-2
ODT command, 3-40
ODT – see Octal Debugging Technique
Off-page references, PAL8, 5-31
OLD command, BASIC, 6-4
Operands, PAL8, 5-10
Operate microinstructions, 5-22
Operators, BASIC,
  arithmetic, 6-10
  relational, 6-12
Operators, FORTRAN IV,
  arithmetic, 7-30
  logical, 7-32
  relational, 7-32
Operators, PAL8, 5-15
Options, 2-7
OR, Boolean inclusive (PAL8), 5-16
OR group skip instructions, PAL8, 5-24
.OR. (logical operator), FORTRAN IV, 7-32
Output,
  COMPARE, 3-4
  CREF, 3-15, 5-17
  MAP, 3-36
Output file extensions, FORTRAN IV, 7-4
Output files,
  BATCH, 8-1
  Editor, 4-7
  FORTRAN IV, 7-5
Output specifications,
  BATCH, 8-1
  command, 2-6
  wildcards, 2-10

Page format control, PAL8, 5-28
Page makeup, Editor, 4-1
Page zero addressing, PAL8, 5-25
PAL command, 3-41
PAL8 Assembler,
  calling, 5-1
  characters, 5-9, 5-10
  error messages, 5-32
  instruction set, 5-21
  link generation and storage, 5-31
  options, 5-8
  permanent symbol table, 5-34
  program preparation, 5-4
  pseudo-operators, 5-25
  statements, 5-10
  terminating assembly, 5-32
Parentheses,
  BASIC arithmetic operations, 6-11
  input/output options, 2-8
  PAL8, 5-19
PAUSE statement, FORTRAN IV, 7-42
Period (.) character – see Dot (.) character
Permanent device names, keyboard monitor, 2-3
Permanent symbols, PAL8, 5-12
Permanent symbol table, PAL8, 5-34
PNT (x) function, BASIC, 6-57
POS function, BASIC, 6-48
Postdeletion, 3-9
Predeletion, 3-9
PRINT statement, BASIC, 6-16
PRINT #statement, BASIC, 6-37
Priority of arithmetic operators, BASIC, 6-10
Program assembly, PAL8, 5-1
Program correction, BASIC, 6-7

INDEX (Cont.)

Program execution, BASIC, 6-6
Program termination statements, BASIC, 6-28
Pseudo-operators,
  PAL8, 5-25
  PAL8 conditional, 5-29
  PAL8 nested, 5-29

Question mark (?) in ODT, 9-2
  wild character, 2-9

R command, 3-42
Radix control, PAL8, 5-29
RANDOMIZE statement, BASIC, 6-35
Random number function, BASIC, 6-43
Range,
  DO loop, FORTRAN IV, 7-39
READ statement,
  BASIC, 6-18
  FORTRAN IV, 7-51
Real constants, FORTRAN IV, 7-28
Relational operators,
  BASIC, 6-12
  FORTRAN IV, 7-32
Relocatable binary files, FORTRAN IV, 7-8
RELOC pseudo-op (relocation), PAL8, 5-27
REM statement, BASIC, 6-30
RENAME command, 3-43
RESEQ program, BASIC, 6-7
Reserving memory, PAL8, 5-27
RESTORE statement, BASIC, 6-33
RESTORE #statement, BASIC, 6-40
Restrictions, BATCH, 8-7
RETURN key, 2-1
  BASIC, 6-7
RETURN statement,
  BASIC, 6-28
  FORTRAN IV, 7-62
RETURN subroutine, BASIC, 6-26
REWIND statement, FORTRAN IV, 7-54
RND function, BASIC, 6-43
RUN and RUNNH commands, BASIC, 6-6
RUN command, 3-44
Run-time options, FORTRAN IV, 7-4
Run-time system,
  BASIC, 6-1
  FRTS, 7-10

SAVE command, 3-45
SAVE command, BASIC, 6-5
Scale factor, FORTRAN IV, 7-46
SCRATCH command, BASIC, 6-6
Search mode, Editor, 4-11
SEG$ function, BASIC, 6-48
Semicolon use,
  BASIC format control character, 6-14
  PAL8 statement terminator, 5-11
SET command, 3-48
SGN function, BASIC, 6-44
SIN function,
  BASIC, 6-45
  FORTRAN IV, 7-21
Single character search, Editor, 4-11
Single quotes (') in Hollerith conversion,
    FORTRAN IV, 7-48
Skip instructions, PAL8, 5-24
Slash (/), PAL8 comment field, 5-10
Software, OS/78, 1-1
Source language, FORTRAN IV, 7-27
Source program,
  preparation, FORTRAN IV, 7-1
Space character, PAL8, 5-17
Special characters, PAL8, 5-18
Specification statements, FORTRAN IV, 7-54
Spool device files, BATCH, 8-5
SQRT function,
  BASIC, 6-44
  FORTRAN IV, 7-22
Square brackets ([ ]) characters, PAL8, 5-19
  input/output options, 2-8
Square root function, BASIC, 6-44
SQUISH command, 3-54
START command, 3-53
Statement label, PAL8, 5-10
Statement numbers,
  BASIC, 6-15
  FORTRAN IV, 7-27
  FORTRAN IV internal, 7-5
Statements, BASIC, 6-15 to 6-42
Statements, FORTRAN IV,
  BLOCK DATA, 7-62
  CALL, 7-62
  COMMON, 7-55
  CONTINUE, 7-41
  DATA, 7-58
  DEFINE FILE, 7-50
  DIMENSION, 7-54
  DO, 7-39
  END, 7-42
  EQUIVALENCE, 7-56
  EXTERNAL, 7-63

INDEX (Cont.)

Statements, FORTRAN IV (Cont.),
  FORMAT, 7-42
  FUNCTION, 7-60
  GO TO, 7-36
  IF, 7-38
  PAUSE, 7-42
  READ, 7-51
  RETURN, 7-62
  STOP, 7-42
  SUBROUTINE, 7-61
  WRITE, 7-52
Statements, descriptive, FORTRAN IV,
  arithmetic, 7-34
  assignment, 7-34
  carriage control, 7-48
  control, 7-36
  data transmission, 7-42
  device control, 7-54
  input/output, 7-50
  specification, 7-54
  subprogram, 7-59
  summary, 7-63
  type declaration, 7-59
Statements, PAL8, 5-10
STOP statement,
  BASIC, 6-28
  FORTRAN IV, 7-42
Storage specification, FORTRAN IV, 7-54
STR$ function, BASIC, 6-49
Strings, BASIC,
  array table, 6-32
  character set, 6-12
  concatenation, 6-14
  conventions, 6-13
SUBMIT command, 3-55
Subprograms, FORTRAN IV, 7-24
Subroutines, BASIC, 6-26
  nested, 6-27
Subroutines, FORTRAN IV, 7-16
SUBROUTINE statement, FORTRAN IV, 7-61
Subscripted variables, BASIC, 6-10
Subscripts,
  BASIC, 6-31
  FORTRAN IV, 7-30
Subtraction,
  BASIC, 6-10
  PAL8, 5-15
Suppression of
  listing, PAL8, 5-27
  printed error messages, FORTRAN IV, 7-6

Symbolic instructions, PAL8, 5-15
Symbolic operands, PAL8, 5-15
Symbol map output file, FORTRAN IV loader, 7-8
Symbols, PAL8, 5-12
Symbol table, PAL8, 5-13
System conventions, keyboard monitor, 2-2

TAB function, BASIC, 6-49
Tabs, PAL8, 5-11
TAN function, FORTRAN IV, 7-22
TERMINATE command, 3-57
Termination of assembly, PAL8, 5-32
Terminators, PAL8 statement, 5-11
Text buffer, Editor, 4-1
Text mode, Editor, 4-2
TEXT pseudo-op, 5-30
Text strings, PAL8, 5-30
TRC(x) function, BASIC, 6-52
.TRUE. (logical value), FORTRAN IV, 7-34
Truncation, FORTRAN IV, 7-17
Truth table for logical expressions, FORTRAN IV,
    7-34
Two's complement addition and subtraction,
    PAL8, 5-15
TYPE command, 3-58
Type declaration statements, FORTRAN IV, 7-59
Type, variables, FORTRAN IV, 7-30

UA, UB, UC commands, 3-60
Unconditional GOTO, FORTRAN IV, 7-36
Underline, ODT, 9-4
User-defined functions, BASIC, 6-34
User defined symbols, PAL8, 5-12
User service routine, C-1

VAL function, BASIC, 6-50
Variable length files, BASIC, 6-36
Variables,
  BASIC, 6-9,
  FORTRAN IV, 7-29

Wild card construction, 2-9
WRITE statement, FORTRAN IV, 7-52

.XOR (logical operator), FORTRAN IV, 7-32

ZERO command, 3-61
Zeroes, leading/trailing FORTRAN IV, 7-28

READER'S COMMENTS

NOTE: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. Problems with software should be reported on a Software Performance Report (SPR) form. If you require a written reply and are eligible to receive one under SPR service, submit your comments on an SPR form.

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 ______________________

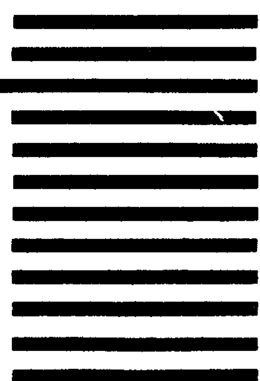
Street ______________________

City ______________ State ______________ Zip Code or Country ______________

Please cut along this line.

Fold Here

Do Not Tear - Fold Here and Staple



BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

Postage will be paid by:

digital

Software Documentation
146 Main Street ML 5-5/E39
Maynard, Massachusetts 01754