March 1980

This document describes how to use the RT-11 operating system. It provides the information required to perform ordinary tasks such as program development, program execution, and file maintenance.

RT-11
System User's Guide

Order No. AA-5279B-TC

SUPERSESSION/UPDATE INFORMATION: This manual supersedes the RT-11 System User's Guide, Order No. DEC-11-ORGDA-A-D, DN1, published March 1978.

OPERATING SYSTEM AND VERSION: RT-11 V4.0

SOFTWARE VERSION: RT-11 V4.0

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, March 1980

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

The following are trademarks of Digital Equipment Corporation:

DIGITAL	DECsystem-10	MASSBUS
DEC	DECtape	OMNIBUS
PDP	DIBOL	OS/8
DECUS	EDUSYSTEM	PHA
UNIBUS	FLIP CHIP	RSTS
COMPUTER LABS	FOCAL	RSX
COMTEX	INDAC	TYPESET-8
DDT	LAB-8	TYPESET-11
DECCOMM	DECSYSTEM-20	TMS-11
ASSIST-11	RTS-8	ITPS-10
VAX	VMS	SBI
DECnet	IAS	PDT
DATATRIEVE	TRAX	

Contents

Page

Preface ..... *xxi*

Part I RT–11 Overview

Chapter 1 System Components 1–1

1.1 Hardware 1–1
1.2 Software 1–2

1.2.1 Monitors 1–2

1.2.1.1 Single-Job (SJ) Monitor 1–3
1.2.1.2 Foreground/Background (FB) Monitor 1–3
1.2.1.3 Extended Memory (XM) Monitor 1–4

1.2.2 Device Handlers 1–5
1.2.3 System Utility Programs 1–5

1.2.3.1 Editing 1–5
1.2.3.2 General Purpose 1–6
1.2.3.3 System Jobs 1–7
1.2.3.4 Debugging and Patching 1–7
1.2.3.5 BATCH 1–8

1.2.4 Language Processors 1–8

1.3 RT–11 Software Documentation 1–9
1.4 Obtaining System Services 1–9

1.4.1 Using the Keyboard Monitor Commands 1–9
1.4.2 Using Programs Directly 1–10
1.4.3 The Relationship Between Complex Commands and System Programs 1–10
1.4.4 The System Macro Library and Programmed Requests 1–11
1.4.5 SYSLIB FORTRAN-Callable Subprograms 1–11

Chapter 2 Program Development 2–1

2.1 Using an Editor 2–1
2.2 Using the Assembler 2–1
2.3 Using the Linker 2–2
2.4 Using the Debugger 2–2
2.5 Using the Librarian 2–2
2.6 Using a High-Level Language 2–3

Page

Part II System Communication

Chapter 3 System Conventions 3–1

3.1 Startup Procedure 3–1
3.2 Data Formats 3–2
3.3 Physical Device Names 3–3
3.4 File Names and File Types 3–4
3.5 Device Structures 3–5
3.6 Special Function Keys 3–6
3.7 Foreground/Background Terminal I/O 3–8
3.8 Type-Ahead Feature 3–9

Chapter 4 Keyboard Commands 4–1

4.1 Command Syntax 4–1

4.1.1 Factoring File Specifications 4–3
4.1.2 File Type Specifications 4–5
4.1.3 Abbreviating Keyboard Commands 4–5
4.1.4 Keyboard Prompts 4–6

4.2 Wildcards 4–7

4.3 Indirect Files 4–9

4.3.1 Creating Indirect Files 4–9
4.3.2 Executing Indirect Files 4–12
4.3.3 Startup Indirect Files 4–15

4.4 Keyboard Monitor Commands 4–15
APL 4–17
ASSIGN 4–18
B 4–20
BASIC 4–21
BOOT 4–22
CLOSE 4–24
COMPILE 4–25
COPY 4–32
CREATE 4–44
D 4–46
DATE 4–47
DEASSIGN 4–48
DELETE 4–49
DIBOL 4–52
DIFFERENCES 4–56
DIRECTORY 4–62
DUMP 4–73
E 4–78
EDIT 4–79
EXECUTE 4–82
FORMAT 4–90
FORTRAN 4–93
FRUN 4–99

Page

GET 4–102
GT 4–103
HELP 4–105
INITIALIZE 4–107
INSTALL 4–112
LIBRARY 4–113
LINK 4–119
LOAD 4–126
MACRO 4–128
PRINT 4–133
R 4–137
REENTER 4–138
REMOVE 4–139
RENAME 4–140
RESET 4–143
RESUME 4–144
RUN 4–145
SAVE 4–147
SET 4–149
SHOW 4–160
SQUEEZE 4–167
SRUN 4–169
START 4–171
SUSPEND 4–172
TIME 4–173
TYPE 4–174
UNLOAD 4–176

Part III Text Editing

Chapter 5 Text Editor (EDIT) 5–1

5.1 Calling EDIT 5–1
5.2 Modes of Operation 5–1
5.3 Special Key Commands 5–2
5.4 Command Structure 5–3

5.4.1 Arguments 5–5
5.4.2 Command Strings 5–6
5.4.3 Current Location Pointer 5–7
5.4.4 Character- and Line-Oriented Command Properties 5–7
5.4.5 Command Repetition 5–9

5.5 Memory Usage 5–10
5.6 Editing Commands 5–12

5.6.1 File Open and Close Commands 5–12

5.6.1.1 Edit Read 5–12
5.6.1.2 Edit Write 5–13
5.6.1.3 Edit Backup 5–13
5.6.1.4 End File 5–14

Page

5.6.2 File Input/Output Commands 5–15
5.6.2.1 Read 5–14
5.6.2.2 Write 5–16
5.6.2.3 Next 5–17
5.6.2.4 EXit 5–18
5.6.3 Pointer Relocation Commands 5–19
5.6.3.1 Beginning 5–19
5.6.3.2 Jump 5–20
5.6.3.3 Advance 5–20
5.6.4 Search Commands 5–21
5.6.4.1 Get 5–21
5.6.4.2 Find 5–23
5.6.4.3 Position 5–23
5.6.5 Text Listing Commands 5–24
5.6.5.1 List 5–24
5.6.5.2 Verify 5–25
5.6.6 Text Modification Commands 5–25
5.6.6.1 Insert 5–25
5.6.6.2 Delete 5–26
5.6.6.3 Kill 5–27
5.6.6.4 Change 5–28
5.6.6.5 Exchange 5–30
5.6.7 Utility Commands 5–30
5.6.7.1 Save 5–30
5.6.7.2 Unsave 5–31
5.6.7.3 Macro 5–32
5.6.7.4 Execute Macro 5–33
5.6.7.5 Edit Version 5–33
5.6.7.6 Upper- and Lower-Case Commands 5–34
5.7 Display Editor 5–34
5.7.1 Using the Display Editor 5–35
5.7.2 Immediate Mode 5–37
5.8 EDIT Example 5–38
5.9 EDIT Error Conditions 5–39

Part IV Utility Programs

Chapter 6 Command String Interpreter 6–1

6.1 Command String Interpreter Syntax 6–1
6.2 Prompting Characters 6–3

Page

Chapter 7 Peripheral Interchange Program (PIP) 7–1

7.1 Calling PIP 7–1
7.2 Options 7–3

7.2.1 Operations Involving Magtape and Cassette 7–4
7.2.1.1 Using Cassette 7–4
7.2.1.2 Using Magtape 7–8
7.2.2 Copy Operations 7–10
7.2.2.1 Image Mode 7–10
7.2.2.2 ASCII Mode (/A) 7–11
7.2.2.3 Binary Mode (/B) 7–11
7.2.2.4 Newfiles Option (/C) 7–11
7.2.2.5 Ignore Errors Option (/G) 7–12
7.2.2.6 Copies Option (/K:n) 7–12
7.2.2.7 Noreplace Option (/N) 7–12
7.2.2.8 Predelete Option (/O) 7–12
7.2.2.9 Exclude Option (/P) 7–13
7.2.2.10 Single-Block Transfer Option (/S) 7–13
7.2.2.11 Setdate Option (/T) 7–13
7.2.2.12 Concatenate Option (/U) 7–13
7.2.2.13 System Files Option (/Y) 7–13
7.2.3 Delete Option (/D) 7–14
7.2.4 Rename Option (/R) 7–14
7.2.4.1 File Protection Option (/F) 7–15
7.2.4.2 File "Unprotection" Option (/Z) 7–15
7.2.5 Logging Option (/W) 7–15
7.2.6 Query Option (/Q) 7–15
7.2.7 Wait Option (/E) 7–16
7.2.7.1 Single-Volume Operation 7–16
7.2.7.2 Double-Volume Operation 7–17

Chapter 8 Device Utility Program (DUP) 8–1

8.1 Calling DUP 8–1
8.2 Options 8–1

8.2.1 Create Option (/C/G:n) 8–3
8.2.2 Image Copy Option (/I) 8–4
8.2.3 Bad Block Scan Option (/K) 8–6
8.2.3.1 File Option (/F) 8–7
8.2.4 Boot Option (/O) 8–8
8.2.5 Boot Foreign Volume Option (/Q) 8–9
8.2.6 Squeeze Option (/S) 8–9
8.2.7 Extend Option (/T:n) 8–10
8.2.8 Bootstrap Copy Option (/U) 8–11
8.2.9 Volume ID Option (/V[:ONL]) 8–12
8.2.10 Wait for Volume Option(/W) 8–13
8.2.11 Noquery Option (/Y) 8–13
8.2.12 Directory Initialization Option (/Z[:n]) 8–14

Page

8.2.12.1 Changing Directory Segments (/N:n) 8–14
8.2.12.2 Storing Volume ID (/V) 8–15
8.2.12.3 Replacing Bad Blocks (/R[:RET]) 8–15
8.2.12.4 Covering Bad Blocks (/B) 8–16
8.2.12.5 Restoring a Disk (/D) 8–17

Chapter 9 Directory Program (DIR) 9–1

9.1 Calling DIR 9–1
9.2 Options 9–2

9.2.1 Alphabetical Option (/A) 9–3
9.2.2 Block Number Option (/B) 9–4
9.2.3 Columns Option (/C:n) 9–4
9.2.4 Date Option (/D[:date]) 9–4
9.2.5 Entire Option (/E) 9–5
9.2.6 FAST Option (/F) 9–5
9.2.7 Begin Option (/G) 9–5
9.2.8 Since Option (/J[:date]) 9–6
9.2.9 Before Option (/K[:date]) 9–6
9.2.10 Listing Option (/L) 9–6
9.2.11 Unused Areas Option (/M) 9–7
9.2.12 Summary Option (/N) 9–7
9.2.13 Octal Option (/O) 9–7
9.2.14 Exclude Option (/P) 9–8
9.2.15 Deleted Option (/Q) 9–8
9.2.16 Reverse Option (/R) 9–9
9.2.17 Sort Option (/S[:xxx]) 9–9
9.2.18 Volume ID Option (/V[:ONL]) 9–10

Chapter 10 MACRO–11 Program Assembly 10–1

10.1 Calling the MACRO–11 Assembler 10–1
10.2 Terminating the MACRO–11 Assembler 10–3
10.3 Temporary Work File 10–3
10.4 File Specification Options 10–4

10.4.1 Listing Control Options 10–5
10.4.2 Function Control Options 10–7
10.4.3 Macro Library File Designation Option 10–8
10.4.4 Cross-Reference (CREF) Table Generation Option ... 10–9

10.4.4.1 Obtaining a Cross-Reference Table 10–9
10.4.4.2 Handling Cross-Reference Table Files 10–10

10.4.5 Assembly Pass Option 10–13

10.5 MACRO–11 8K Version 10–13
10.6 MACRO–11 Error Codes 10–14

Page

Chapter 11 Linker (LINK) 11–1

11.1 Overview of the Linker Process 11–1

11.1.1 What the Linker Does 11–2
11.1.1 How the Linker Structures the Load Module 11–3

11.1.2.1 Absolute Section 11–3
11.1.2.2 Program Sections 11–3

11.1.3 Global Symbols: Communication Links Between Modules 11–6

11.2 Calling and Using the Linker 11–7
11.3 Input and Output 11–11

11.3.1 Input Object Modules 11–11
11.3.2 Input Library Modules 11–11
11.3.3 Output Load Module 11–14
11.3.4 Output Load Map 11–16

11.4 Creating an Overlay Structure 11–18

11.4.1 Low Memory Overlays 11–18
11.4.2 Extended Memory Overlays 11–27

11.4.2.1 Virtual Address Space 11–27
11.4.2.2 Physical Address Space 11–28
11.4.2.3 Virtual and Privileged Jobs 11–30
11.4.2.4 Extended Memory Overlay Option (V/:n[:m]) 11–31
11.4.2.5 Combining Low Memory Overlays with Extended Memory Overlays 11–34

11.4.3 Load Map 11–35

11.5 Option Descriptions 11–40

11.5.1 Alphabetical Option (/A) 11–40
11.5.2 Bottom Address Option (/B:n) 11–40
11.5.3 Continue Option (/C) or (//) 11–41
11.5.4 Extend Program Section Option (/E:n) 11–41
11.5.5 Default FORTRAN Library Option (/F) 11–42
11.5.6 Directory Buffer Size Option (/G) 11–42
11.5.7 Highest Address Option (/H:n) 11–42
11.5.8 Include Option (/I) 11–43
11.5.9 Memory Size Option (/K:n) 11–43
11.5.10 LDA Format Option (/L) 11–44
11.5.11 Modify Stack Address Option (/M[:n]) 11–44
11.5.12 Low Memory Overlay Option (/O:n) 11–44
11.5.13 Library List Size Option (/P:n) 11–46
11.5.14 Absolute Base Address Option (/Q) 11–46
11.5.15 REL Format Option (/R[:n]) 11–47
11.5.16 Symbol Table Option (/S) 11–47
11.5.17 Transfer Address Option (/T[:n]) 11–47

Page

11.5.18 Round Up Option (/U:n) 11–48
11.5.19 Extended Memory Overlay Option (/V:n[:m]) 11–49
11.5.20 Map Width Option (/W) 11–49
11.5.21 Bitmap Inhibit Option (/X) 11–49
11.5.22 Boundary Option (/Y:n) 11–49
11.5.23 Zero Option (/Z:n) 11–50

11.6 Linker Prompts 11–50

Chapter 12 Librarian (LIBR) 12–1

12.1 Calling and Using LIBR 12–1
12.2 Option Commands and Functions for Object Libraries 12–3

12.2.1 Include All Global Symbols, Even Absolute Global Symbols Option(/A) 12–3
12.2.2 Command Continuation Options (/C and //) 12–4
12.2.3 Creating a Library File 12–4
12.2.4 Inserting Modules into a Library 12–5
12.2.5 Delete Option (/D) 12–6
12.2.6 Extract Option (/E) 12–6
12.2.7 Delete Global Option (/G) 12–7
12.2.8 Include Module Names Option (/N) 12–7
12.2.9 Include P-section Names Option (/P) 12–8
12.2.10 Replace Option (/R) 12–8
12.2.11 Update Option (/U) 12–9
12.2.12 Wide Option (/W) 12–9
12.2.13 Creating Multiple Definition Libraries Option (/X) ... 12–10
12.2.14 Listing the Directory of a Library File 12–10
12.2.15 Merging Library Files 12–11
12.2.16 Combining Library Option Functions 12–12

12.3 Option Commands and Functions for Macro Libraries 12–13

12.3.1 Command Continuation Options (/C or //) 12–13
12.3.2 Macro Option (/M[:n]) 12–14

Chapter 13 Dump Utility (DUMP) 13–1

13.1 Calling and Using DUMP 13–1
13.2 Options 13–1
13.3 Example Commands and Listings 13–2

Chapter 14 File Exchange Program (FILEX) 14–1

14.1 File Formats 14–1
14.2 Calling and Using FILEX 14–2
14.3 Options 14–2

14.3.1 Transferring Files Between RT–11 and DOS/BATCH (or RSTS) 14–3
14.3.2 Transferring Files Between RT–11 and Interchange Diskette 14–5

Page

14.3.3 Transferring Files to RT–11 from DECsystem- 10 ... 14–8
14.3.4 Listing Directories ... 14–9
14.3.5 Deleting Files from DOS/BATCH (RSTS) DECtapes and Interchange Diskettes ... 14–10

Chapter 15 Source Compare (SRCCOM) ... 15–1

15.1 Calling and Using SRCCOM ... 15–1
15.2 Options ... 15–2
15.3 Output Format ... 15–3

15.3.1 Sample Text ... 15–3
15.3.2 Sample Output Listing ... 15–3
15.3.3 /Changebar Option ... 15–6

15.4 Creating a SLP Command File ... 15–7

15.4.1 /Patch Option ... 15–7
15.4.2 /Audit Trail Option ... 15–7

Chapter 16 Binary Comparison Program (BINCOM) ... 16–1

16.1 Calling and Using BINCOM ... 16–1
16.2 Options ... 16–2
16.3 Output Format ... 16–3
16.4 Examples ... 16–4
16.5 Creating a SIPP Command File ... 16–4

Chapter 17 Resource Utility Program (RESORC) ... 17–1

17.1 Calling and Using RESORC ... 17–1
17.2 Options ... 17–1

17.2.1 All Option (/A) ... 17–2
17.2.2 Software Configuration Option (/C) ... 17–3
17.2.3 Device Handler Status Option (/D) ... 17–3
17.2.4 Hardware Configuration Option (/H) ... 17–4
17.2.5 Loaded Jobs Option (/J) ... 17–5
17.2.6 Device Assignments Option (/L) ... 17–5
17.2.7 Current Monitor Option (/M) ... 17–6
17.2.8 Special Features Option (/O) ... 17–7
17.2.9 Terminal Status Option (/T) ... 17–8
17.2.10 Summary Option (/Z) ... 17–9

Chapter 18 Volume Formatting Program (FORMAT) ... 18–1

18.1 Calling and Using FORMAT ... 18–1
18.2 Options ... 18–2

18.2.1 Default Format ... 18–3
18.2.2 Pattern Verification Option (/P:n) ... 18–3
18.2.3 Single-Density Option (/S) ... 18–5

Page

18.2.4 Verification Option (/V[:ONL]) 18–5
18.2.5 Wait Option (/W) 18–5
18.2.6 Noquery Option (/Y) 18–5

Part V System Jobs

Chapter 19 Error Logging 19–1

19.1 Uses 19–1
19.2 Error Logging Subsystem 19–2
19.3 Calling and Using the Error Logger 19–3

19.3.1 Using ELINIT 19–4
19.2.2 Using ERROUT 19–5

19.4 Report Analysis 19–6

19.4.1 Storage Device Error Report 19–6
19.4.2 Memory Error Report 19–8
19.4.3 Summary Error Report 19–9

Chapter 20 Queue Package 20–1

20.1 Calling and Using the Queue Package 20–1

20.1.1 Calling and Running QUEUE 20–1
20.1.2 Calling and Running QUEMAN 20–2

20.2 QUEMAN Options 20–3

20.2.1 Halting QUEUE (/A) 20–4
20.2.2 Deleting Files After Printing (/D) 20–4
20.2.3 Printing Banner Pages (/H:n) 20–4
20.2.4 Printing Multiple Copies (/K:n) 20–5
20.2.5 Removing a Job from the Queue (/M) 20–5
20.2.6 Listing the Contents of the Queue (/L) 20–5
20.2.7 No Banner Pages Option (/N) 20–6
20.2.8 Setting the Queue Package Defaults (/P) 20–6
20.2.9 Suspending Output (/S) 20–7
20.2.10 Resuming/Restarting Output (/R) 20–7
20.2.11 Continuing a Command String (//) 20–8

Part VI Debugging and Altering Programs

Chapter 21 On-Line Debugging Technique (ODT) 21–1

21.1 Calling and Using ODT 21–1
21.2 Relocation 21–4
21.3 Commands and Functions 21–6

21.3.1 Printout Formats 21–6
21.3.2 Opening, Changing, and Closing Locations 21–6

Page

21.3.2.1 Slash (/) ... 21–7
21.3.2.2 Backslash (\) ... 21–7
21.3.2.3 LINE FEED Key (LF) ... 21–8
21.3.2.4 The Circumflex or Up Arrow (^) ... 21–8
21.3.2.5 Underline or Back-Arrow (_) ... 21–8
21.3.2.6 Open the Addressed Location (@) ... 21–9
21.3.2.7 Relative Branch Offset (>) ... 21–9
21.3.2.8 Return to Previous Sequence (<) ... 21–9

21.3.3 Accessing General Registers 0-7 ... 21–9
21.3.4 Accessing Internal Registers ... 21–10
21.3.5 Radix-50 Mode (X) ... 21–10
21.3.6 Breakpoints ... 21–11
21.3.7 Running the Program (r;G and r;P) ... 21–12
21.3.8 Single-Instruction Mode ... 21–14
21.3.9 Searches ... 21–14

21.3.9.1 Word Search (r;W) ... 21–15
21.3.9.2 Effective Address Search (r;E) ... 21–15

21.3.10 Constant Register (r;C) ... 21–16
21.3.11 Memory Block Initialization (;F and ;I) ... 21–16
21.3.12 Calculating Offsets (r;O) ... 21–17
21.3.13 Relocation Register Commands ... 21–18
21.3.14 The Relocation Calculators, n! and nR ... 21–19
21.3.15 ODT Priority Level/($P) ... 21–19
21.3.16 ASCII Input and Output (r;nA) ... 21–20

21.4 Programming Considerations ... 21–21

21.4.1 Using ODT with Foreground/Background Jobs ... 21–21
21.4.2 Functional Organization ... 21–22
21.4.3 Breakpoints ... 21–22
21.4.4 Searches ... 21–25
21.4.5 Terminal Interrupt ... 21–26

21.5 Error Detection ... 21–26

Chapter 22 Save Image Patch Program (SIPP) ... 22–1

22.1 Calling and Using SIPP ... 22–1
22.2 SIPP Options ... 22–2
22.3 SIPP Dialog ... 22–3
22.4 SIPP Commands ... 22–4

22.4.1 Opening and Modifying Locations Within a File ... 22–6
22.4.2 Backing Up Through Files ... 22–6
22.4.3 Advancing in Bytes ... 22–6
22.4.4 Entering Octal Values (;O) ... 22–7
22.4.5 Displaying and entering ASCII Values ... 22–7
22.4.6 Displaying and entering RAD50 Values ... 22–8
22.4.7 Searching Through Files (;S) ... 22–9
22.4.8 Verifying (;V) ... 22–10

Page

22.4.9 Backing Up to a Previous Prompt 22–11
22.4.10 Completing Code Modifications 22–12
22.4.11 Extending Files and Overlay Segments 22–12
22.4.11.1 Non overlaid Program 22–13
22.4.11.2 Overlaid Program, Low Memory Overlays Only 22–13
22.4.11.3 Overlaid Program, Extended Memory Overlays Only 22–14
22.4.11.4 Overlaid Program, Both Low Memory and Extended Memory Overlays 22–14

22.5 SIPP Checksum 22–15
22.6 Running SIPP from an Indirect File 22–16
22.7 Running SIPP from a Batch Stream 22–17

Chapter 23 Object Module Patch Utility (PAT) 23–1

23.1 Calling and Using PAT 23–1
23.2 How PAT Effects Updates 23–3
23.2.1 Input File 23–4
23.2.2 Correction File 23–4
23.3 Updating Object Modules 23–5
23.3.1 Overlaying Lines in a Module 23–5
23.3.2 Adding a Subroutine to a Module 23–6
23.4 Determining and Validating the Contents of a File 23–8

Chapter 24 Source Language Patch Program (SLP) 24–1

24.1 Calling and Using SLP 24–1
24.2 Options 24–2
24.3 Example 24–3
24.4 Creating and Maintaining a Command File 24–4
24.4.1 Update Line Format 24–4
24.4.2 Creating a Numbered Listing 24–6
24.4.3 Adding Lines to a File 24–6
24.4.4 Deleting Lines in a File 24–8
24.4.5 Replacing Lines in a File 24–9

Chapter 25 Patch Utility (PATCH) 25–1

25.1 Calling and Using PATCH 25–1
25.2 Options 25–1
25.2.1 Checksum 25–2
25.3 Commands 25–2

Page

25.3.1 Patching a New File (F) ... 25–4
25.3.2 Exiting from PATCH (E) ... 25–4
25.3.3 Examining and Changing Locations in the File ... 25–4
25.3.4 Translating and Indirectly Modifying Locations with a File ... 25–5
25.3.5 Setting Values in the Overlay Handler Tables of a Program ... 25–7
25.3.6 Including the Old Contents into the Checksum ... 25–7
25.3.7 Setting the Bottom Address ... 25–7
25.3.8 Setting Relocation Registers ... 25–8

25.4 Examples ... 25–8

25.4.1 Patching a Non Overlaid File ... 25–8
25.4.2 Patching an Overlaid File ... 25–10

Appendix A BATCH ... A–1

A.1 Hardware and Software Requirements ... A–1
A.2 Control Statement Format ... A–2

A.2.1 Command Fields ... A–2

A.2.1.1 Command Names ... A–2
A.2.1.2 Command Field Options ... A–3

A.2.2 Specification Fields ... A–5

A.2.2.1 Physical Device Names ... A–5
A.2.2.2 File Specifications ... A–6
A.2.2.3 Wildcard Construction ... A–6
A.2.2.4 Specification Field Options ... A–7

A.2.3 Comment Fields ... A–7
A.2.4 BATCH Character Set ... A–8
A.2.5 Temporary Files ... A–9

A.3 General Rules and Conventions ... A–10
A.4 Commands ... A–11

A.4.1 $BASIC ... A–12
A.4.2 $CALL ... A–13
A.4.3 $CHAIN ... A–14
A.4.4 $COPY ... A–15
A.4.5 $CREATE ... A–15
A.4.6 $DATA ... A–16

A.4.6.1 Using $DATA with FORTRAN Programs ... A–17

A.4.7 $DELETE ... A–18
A.4.8 $DIRECTORY ... A–18
A.4.9 $DISMOUNT ... A–18
A.4.10 $EOD ... A–19
A.4.11 $EOJ ... A–20

Page

A.4.12 $FORTRAN A–20
A.4.13 $JOB A–22
A.4.14 $LIBRARY A–23
A.4.15 $LINK A–24
A.4.16 $MACRO A–26
A.4.17 $MESSAGE A–28
A.4.18 $MOUNT A–29
A.4.19 $PRINT A–31
A.4.20 $RT11 ..... A–31
A.4.21 $RUN A–31
A.4.22 $SEQUENCE A–32
A.4.23 Sample BATCH Stream A–32

A.5 RT–11 Mode A–34

A.5.1 Communicating with RT–11 ..... A–35
A.5.2 Creating RT–11 Mode BATCH Programs A–36

A.5.2.1 Labels A–36
A.5.2.2 Variables A–37
A.5.2.3 Terminal I/O Control A–39
A.5.2.4 Other Control Characters A–39
A.5.2.5 Comments A–40

A.5.3 RT–11 Mode Examples A–40

A.6 Creating BATCH Programs on Punched Cards A–41
A.7 Operating Procedures A–42

A.7.1 Loading BATCH A–42
A.7.2 Running BATCH A–44
A.7.3 Communicating with BATCH Jobs A–47
A.7.4 Terminating BATCH A–50

A.8 Differences Between RT–11 BATCH and RSX–11D BATCH A–50

Appendix B Monitor Command Abbreviations and System Utility Program Equivalents

Index

Figures

2-1 Program Development Cycle 2–3
4-1 Sample Command Syntax Illustration 4–3
4-2 Format of a 12-bit Binary Number 4–149
5-1 Display Editor Format, 12-inch Screen 5–35
10-1 Sample Assembly Listing 10–5
10-2 Cross-Reference Table 10–12
11-1 Library Searches 11–13
11-2 Sample Load Map 11–17
11-3 Sample Overlay Structure for a FORTRAN Program 11–19
11-4 Overlay Scheme 11–20
11-5 The Run-Time Overlay Handler 11–20
11-6 Sample Subroutine Calls and Return Paths 11–23

		Page
11-7	Memory Diagram Showing BASIC Link with Overlay Regions	11–26
11-8	Program Virtual Address Space	11–28
11-9	Physical Address Space for Program with Low Memory Overlays	11–29
11-10	Virtual and Physical Address Space	11–30
11-11	Virtual and Physical Address Space	11–32
11-12	Extended Memory Partitions That Contain Sharing Segments	11–33
11-13	Memory Diagram Showing Low Memory and Extended Memory Overlays	11–35
11-14	Load Map for Program with Unmapped and Virtual Overlays	11–36
11-15	Extended Memory Overlay Handler	11–38
19-1	Error Logging Subsystem	19–3
19-2	Sample Storage Device Error Report	19–7
19-3	Sample Memory Parity Error Report	19–8
19-4	Sample Cache Memory Error Report	19–8
19-5	Sample Summary Error Report for Device Statistics	19–9
19-6	Sample Summary Error Report for Memory Statistics	19–10
19-7	Sample Report File Environment and Error Count Report	19–10
21-1	Linking ODT with a Program	21–2
23-1	Updating a Module Using PAT	23–2
23-2	Processing Steps Required to Update a Module Using PAT	23–2
A-1	EOF Card	A–42

Tables

1-1	RT–11 Hardware Components	1–1
3-1	Permanent Device Names	3–3
3-2	Standard File Types	3–4
3-3	Device Structures	3–6
3-4	Special Function Keys	3–7
4-1	Commands Supporting Wildcards	4–8
4-2	Wildcard Defaults	4–8
4-3	Sort Categories	4–69
4-4	Optimization Codes	4–96
4-5	FORTRAN Listing Codes	4–97
4-6	Display Screen Values	4–104
4-7	Default Directory Sizes	4–110
4-8	Execution and Prompting Sequence of LIBRARY Options	4–117
4-9	Prompting Sequence for LINK Options	4–120
4-10	Cross-reference Sections	4–129
4-11	.DSABL and .ENABL Directive Summary	4–129
4-12	.LIST and .NLIST Directive Summary	4–132
4-13	SET Device Conditions and Modification	4–150
5-1	EDIT Key Commands	5–2
5-2	EDIT Command Categories	5–4
5-3	Command Arguments	5–5
5-4	EDIT Commands and File Status	5–15
5-5	Write Command Arguments	5–17
5-6	Jump Command Arguments	5–20

		Page
5-7	Advance Command Arguments	5–21
5-8	List Command Arguments	5–24
5-9	Delete Command Arguments	5–27
5-10	Kill Command Arguments	5–28
5-11	Change Command Arguments	5–29
5-12	Exchange Command Arguments	5–30
5-13	Unsave Command Arguments	5–31
5-14	M Command and Arguments	5–32
5-15	Immediate Mode Commands	5–37
6-1	Prompting Characters	6–3
7-1	PIP Options	7–3
8-1	DUP Options and Categories	8–2
8-2	DUP Options	8–2
8-3	Default Directory Sizes	8–15
9-1	DIR Options	9–2
9-2	Sort Codes	9–9
10-1	Default File Specification Values	10–3
10-2	File Specification Options	10–4
10-3	Arguments for /L and /N Options	10–5
10-4	Arguments for /E and /D Options	10–6
10-5	/C Option Arguments	10–10
10-6	MACRO–11 Error Codes	10–14
11-1	P-sect Attributes	11–4
11-2	Section Attributes	11–6
11-3	P-sect Order	11–6
11-4	Global Reference Resolution	11–7
11-5	Linker Defaults	11–8
11-6	Linker Options	11–9
11-7	Absolute Block Parameters Information	11–15
11-8	Line-by-Line Sample Load Map Description	11–17
11-9	Line-by-Line Sample Load Map Description	11–37
11-10	Linker Prompting Sequence	11–50
12-1	LIBR Object Options	12–3
12-2	LIBR Macro Options	12–13
13-1	DUMP Options	13–1
14-1	Supported FILEX Devices	14–1
14-2	FILEX Options	14–3
15-1	SRCCOM Options	15–2
16-1	BINCOM Options	16–2
17-1	RESORC Options	17–2
18-1	FORMAT Options	18–2
18-2	Verification Bit Patterns	18–4
19-1	Line-by-Line Analysis of the Sample Storage Device Error Report	19–7
19-2	Line-by-Line Analysis of the Sample Memory Error Report	19–8
20-1	QUEMAN Options	20–3
21-1	Forms of Relocatable Expressions (r)	21–5
21-2	Internal Registers	21–10
21-3	Radix–50 Terminators	21–11
21-4	Single Instruction Mode Commands	21–14
21-5	ASCII Terminators	21–20

		Page
22-1	SIPP Options	22–2
22-2	SIPP Commands	22–4
22-3	Overlaid Program Segment Limits	22–14
22-4	Overlaid Program Segment Limits	22–14
22-5	Overlaid Program Segment Limits	22–15
24-1	SLP Options	24–2
24-2	SLP Command File Operators	24–4
25-1	PATCH Options	25–2
25-2	PATCH Commands	25–3
25-3	PATCH Control Characters	25–5
A-1	Command Field Options	A–3
A-2	BATCH File Types	A–6
A-3	Specification Field Options	A–7
A-4	Character Explanation	A–8
A-5	BATCH Commands	A–11
A-6	Operator Directives to BATCH Run-Time Handler	A–48
A-7	Differences Between RT–11 and RSX–11D BATCH	A–50
B-1	Monitor Command/System Utility Program Equivalents	B–1
B-2	System Program/Monitor Command Equivalents	B–1

Preface

This manual describes how to use the RT–11 system; it provides enough information for you to perform ordinary tasks such as program development, program execution, and file maintenance. The manual is written for you if you are already familiar with computer software fundamentals and have some experience using RT–11. If you have no RT–11 experience, you should first read the *Introduction to RT–11* before consulting this manual. If you have experience with an earlier release of RT–11 (this is version 4), you should read the *RT–11 System Release Notes* to learn how RT–11 version 4 differs from earlier versions. If you are interested in more sophisticated programming techniques or in system programming, you should read this manual first and then proceed to the *RT–11 Programmer's Reference Manual* and the *RT–11 Software Support Manual.*

The next section, Chapter Summary, briefly describes the chapters in this manual and suggests a reading path to help you use the manual efficiently.

Chapter Summary

Part I, RT–11 Overview, Chapters 1–2, describes the RT–11 operating system in general. It lists the hardware and software components of the RT–11 system, describes the monitors, and explains the program development process with RT–11.

Part II, System Communication, Chapters 3–4, describes system conventions, such as data formats, physical device names, file naming conventions, and special function key commands. It also introduces the keyboard monitor commands that you use interactively to communicate with the monitor and to perform system jobs.

Part III, Text Editing, Chapter 5, describes the RT–11 text editor (EDIT) and shows you how to create and modify files with it.

Part IV, Utility Programs, Chapters 6–18, consists of thirteen chapters that describe the Command String Interpreter (CSI) and the many programs provided with the RT–11 system. These programs include:

PIP	Peripheral Interchange Program
DUP	Device Utility Program
DIR	Directory Program
MACRO	MACRO–11 Assembly Language
LINK	Linker Program
LIBR	Librarian Program
DUMP	Dump Program
FILEX	File Exchange Program
SRCCOM	Source Comparison Program

BINCOM	Binary File Comparison Program
RESORC	Resource Program
FORMAT	Volume Formatting Program

Part V, System Jobs, Chapters 19–20, describes two utilities that can run as foreground or system jobs: Error Logging and Queue Package. Both utilities actually consist of more than one program and a workfile.

Error Logging comprises three programs: EL (either a foreground or system job), ELINIT (a background job), and ERROUT (a background job).

The Queue Package comprises two programs: QUEMAN (a background job), and QUEUE (either a foreground or system job).

Note that both utilities run under the foreground/background or extended memory monitor, and to run them as system jobs, you must enable system job support through the system generation process.

Part VI, Debugging and Altering Programs, Chapters 21–25, describes the five utility programs that permit you to examine and change assembled programs and source files. These utilities are:

ODT	On-line Debugging Technique
SIPP	Save Image Patch Program
PAT	Object Module Patch Program
SLP	Source Language Patch Program
PATCH	Patch Utility

Appendix A describes BATCH processing. Appendix B contains a summary of the keyboard monitor commands, their abbreviations, and their system program equivalents.

Documentation Conventions

A description of the symbolic conventions used throughout this manual follows. Familiarize yourself with these conventions before you continue reading.

Conventions used in this manual:

1. Examples consist of actual computer output wherever possible. Where necessary, user input is in red to distinguish it from computer output.

2. Where necessary, this manual uses the symbol (RET) to represent a carriage return, (LF) to represent a line feed, (SP) for a space, and (TAB) to represent a tab. Unless the manual indicates otherwise, terminate all commands or command strings with a carriage return.

3. Terminal and console terminal are general terms used throughout all RT–11 documentation to represent any terminal device, including DECwriters, displays, and Teletypes*.

*Teletype is a registered trademark of the Teletype Corporation.

4. You produce several characters in system commands by typing a combination of keys concurrently. For example, while holding down the CTRL key, type O to produce the CTRL/O character. Key combinations such as this one are documented as CTRL/O, CTRL/C, and so on.

5. In discussions of command syntax, upper-case letters represent the command name, which you must type. Lower-case letters represent a variable, for which you must supply a value.

   Square brackets ([]) enclose options: you may include the item in brackets, or you may omit it, as you choose.

   The ellipsis symbol (...) represents repetition. You can repeat the item that precedes the ellipsis.

   The hyphen (-) is a continuation character. Use it at the end of a line if you continue a command string to another line.

   This is a typical illustration of command syntax:

   DELETE[/option...] filespec[/option...]

   This example shows that you must type the word DELETE, as shown, and that you can follow it with one or more options of your choice, but none are required. You must then leave a space, and supply a file specification. The file specification can also be followed by one or more options, but none are required. Here is a typical command string:

```
DELETE/NOQUERY/DATE  DTA:MYFILE.FOR
```

Part I
RT–11 Overview

Part I of this manual provides a description of the hardware and software components that make up the RT–11 operating system and a summary of the program development cycle.

Chapter 1 lists all the hardware devices, monitors, utility programs, and language processors available in the RT–11 computer system. This chapter also lists the system services, called keyboard commands, available in RT–11.

Chapter 2 gives a general description of the steps involved in the program development cycle. This chapter also summarizes the use of the RT–11 librarian and high-level languages.

Chapter 1
System Components

RT–11 is DIGITAL's smallest real-time and program development operating system for the PDP–11 family of minicomputers. This single-user operating system runs on hardware configurations ranging from the microprocessor-based PDP–11/03 through the larger PDP–11/60 with cache memory. RT–11's design philosophy is to be small, efficient, reliable, and easy to use.

The RT–11 computer system consists of hardware, software, and documentation. This chapter describes briefly the components available for you to use with RT–11.

1.1 Hardware

The hardware components of an RT–11 system are drawn from the following categories:

- PDP–11 and PDT–11 family computers (except the 11/70 or VAX 11/780)
- Printing and video terminals
- Core and solid state memory
- Line frequency and programmable clocks
- Random-access mass storage devices
- Other peripheral devices

The smallest possible hardware configuration for an RT–11 system must include a PDP–11 or PDT–11 computer, one terminal, 12K words of memory, and a random-access mass storage device. Larger systems can have a clock, more memory, more terminals, and more peripheral devices.

Table 1-1 lists specific hardware devices that can make up an RT–11 computer system.

Table 1–1: RT–11 Hardware Components

Device Type	Controller	Device Name
Card reader	CR11 CM11	CR11 CM11
Cassette	TA11	TU60
Clock		KW11–L, KW11–P

(continued on next page)

Table 1–1: RT–11 Hardware Components (Cont.)

Device Type	Controller	Device Name
DECtape	TC11	TU56
DECtape II data cartridge	DL11, DLV11	TU58
Disk cartridge	RK11, RKV11 RK611 RL11, RLV11	RK05, RK05F RK06, RK07 RL01, RL02
Diskette	RX11, RXV11 RX211, RXV21	RX01 RX02
Fixed-head	RF11 RH11	RS11 RJS03, RJS04
Removable pack	RP11	RP02, RP03
Display processor	VT11 VS60	VR14-L, VR17-L
Line printer	LS11 LV11 LP11, LPV11	LS11, LA180 LV11 (printer only) all LP11-controlled printers
Magtape	TM11, TMA11 RH11 TS11	TU10, TS03, TE16 TJU16, TU45
Paper tape		
Reader	PR11	PR11
Reader/punch	PC11	PC11
Terminal	DL11, DLV11 DZ11, DZV11	LA30P, LA36, LA120, LS120, LT33, LT35, VT05, VT50, VT52, VT55, VT61, VT100
Terminal and clock	DL11–W	terminal/clock combination

1.2 Software

The software components of the RT–11 computer system can be divided into four general groups:

- Monitors
- Device handlers
- Utility programs
- Language processors

These are described in the following sections.

1.2.1 Monitors

An RT–11 monitor is a collection of routines that control the operation of programs, schedule operations, allocate resources, and perform input and output. A monitor comprises three major components: RMON (resident monitor); USR (user service routine); and KMON (keyboard monitor). The resident monitor (RMON) is the part of the monitor that is always present in

memory. It is the executive controller for the entire system. The user service routine (USR) performs operations related to input and output, such as opening and closing files. The keyboard monitor (KMON) is the interface between you and the other parts of the system. It contains routines to process the keyboard monitor commands, which are your means of performing common system operations such as loading and running programs, assigning alternate device names, and copying and deleting files.

RT–11 provides three different operating environments that represent compromises among size, speed, and capability. Three types of monitor, all containing the main parts described above, supervise the different environments. These three monitors are the single-job monitor (SJ), the foreground/background monitor (FB), and the extended memory monitor (XM). The three environments are upward compatible:

- The single-job monitor supports the basic environment.
- The foreground/background monitor includes all the support of the single-job monitor and adds some extra features.
- The extended memory monitor is an extension of the foreground/background monitor that includes all the foreground/background features in addition to the extended memory capabilities.

1.2.1.1 Single-Job (SJ) Monitor – The single-job monitor, called the SJ monitor, can run one job at a time. It is the smallest of the three monitors. While the SJ monitor does not offer some of the optional features that the other monitors have, you can use all the system utility programs, most of the keyboard monitor commands, and many of the programmed requests. Only 12K words of memory are required for a single-job system, though, and since the SJ monitor uses 2K words itself, this leaves 10K words for system utility programs or for your application program. The SJ monitor is ideal for real-time applications that require a high data transfer rate because it services interrupts quickly. In the SJ environment, programs can access up to 28K words of memory (and up to 30K words on some LSI–11's).

A version of the SJ monitor, the base-line (BL) monitor, can run in a minimum configuration of 8K words of memory, but it does not support optional monitor and device functions. The BL monitor is best suited for very small hardware configurations, or for larger configurations where the application requires minimal executive support.

1.2.1.2 Foreground/Background (FB) Monitor – The foreground/background monitor, called the FB monitor, can accommodate two jobs that appear to run concurrently: a foreground job, and a background job. All programs that run in the single-job environment, including system utility programs and language processors, can run as background jobs in the foreground/background environment. The foreground job is the time-critical, real-time job, and the FB monitor gives it priority over the background job. A foreground/background system requires 16K words of memory and a system clock.

Quite often, the central processor of a computer system spends much of its time waiting for some external event to occur. Usually, this event is a real-time interrupt or the completion of an I/O transfer. The FB monitor lets you take advantage of the unused processor capacity to accomplish lower priority tasks in the background.

Whenever the foreground job reaches a state in which no useful processing can be done until some external event occurs, the monitor executes the background job. The background job runs until the foreground job is again ready to execute. The processor then interrupts the background job and resumes the foreground job.

In effect, the FB monitor allows a time-critical job to run in the foreground while a less critical task, takes place in the background. All the system utility programs and language processors can run as background jobs in a foreground/background system. Thus, you can use FORTRAN or EDIT, for example, in the background, while the foreground job is collecting, storing, and analyzing data.

Compared to the SJ monitor, the FB monitor is somewhat larger and has slightly slower response time. However, it provides support for the foreground/background environment. In the FB environment, programs can access 28K words of memory (and up to 30K words on some LSI–11's). Special keyboard monitor commands are designed to link, run, suspend, and resume foreground jobs. In addition, programmed requests permit a foreground job and a background job to share data. Special system jobs (described in Part V) run in the foreground/background environment.

1.2.1.3 Extended Memory (XM) Monitor – The extended memory monitor, called XM, includes all the features of the FB monitor. In fact, it is produced by a conditional assembly of the FB monitor file, RMONFB. Throughout this manual, references to the FB environment also apply to the XM environment, unless otherwise stated. The XM monitor provides means for a program to access up to 124K words of physical memory. It permits foreground and background jobs to extend their logical program space beyond the 32K-word limit imposed by the 16-bit PDP–11 address word. The XM monitor requires a system with the Extended Instruction Set (EIS), a KT11 memory management unit, and more than 28K words of memory.

Extended memory services, or the ability to use memory mapping, are available at a variety of levels. The XM environment was originally designed primarily for high-level language processors such as DIBOL and FORTRAN that automatically make use of extended memory. For DIBOL users, for example, the mapping to extended memory is completely transparent. FORTRAN programmers can use virtual arrays to store large amounts of data in extended memory. Now, a command to the linker permits RT–11 programmers to store overlays in extended memory instead of on disk, thus increasing an overlaid program's execution speed markedly. A .SETTOP programmed request permits a MACRO–11 program to dynamically allocate buffers in extended memory without concern for memory mapping. Finally, on the most basic level, RT–11 provides other programmed requests

that MACRO–11 programs can use to control their own mapping to extended memory. Keep in mind that designing an application program to use extended memory this way requires considerable thought and careful planning.

In the XM environment, jobs are described as being either privileged or virtual jobs. Foreground or background jobs that execute in the FB or SJ environments can also execute in the XM environment as privileged jobs. That is, they use the default mapping from logical virtual to physical memory. Except for jobs that include interrupt service routines, these privileged jobs need no major changes to execute properly in the XM environment. Jobs that you design specifically to take advantage of extended memory are called virtual jobs.

1.2.2 Device Handlers

Device handlers are routines that provide the interface to the various hardware devices that are part of the computer system. The handlers drive, or service, peripheral devices and control the physical activities on the devices. In RT–11, the terms *device handler* and *device driver* are used interchangeably.

A handler exists for every device the system supports (except for the VT–11). When you reference a device by its physical name, such as DL: for the RL01 disk, you are actually referring to the name of the device handler for that peripheral.

Chapter 3 contains a list of all the devices that RT–11 supports, along with their physical names. If you need to use a peripheral device that is not supported by RT–11, you usually must write the handler for it yourself. The procedure for doing this is documented in the *RT–11 Software Support Manual.*

1.2.3 System Utility Programs

RT–11 provides a number of utility programs that are designed to help you develop programs and perform system housekeeping. The following sections describe these utilities briefly and show where they are documented in greater detail.

1.2.3.1 Editing – You use text editors to create and modify source programs and to maintain files of any ASCII data, such as memos or documentation for your own application programs. DIGITAL distributes three text file editors with RT–11, so that you can choose the one that best suits your needs and experience: EDIT, KED, and K52.

The RT–11 text editor (EDIT, described in Chapter 5) is a character-oriented editor suitable for hard copy terminals. Its text manipulation commands

permit you to make text insertions or changes quickly and easily. EDIT also has a special mode for VT11 or VS60 graphics display terminals.

The keypad editor (KED, and K52 described in the *PDP–11 Keypad Editor User's Guide*) is designed especially for the VT100 and VT52 video terminals that have the special function keypad. The keypad keys control the editing functions. They permit you to position a visible cursor anywhere in your text file and make insertions or changes easily. KED runs on VT100 terminals, and K52 runs on VT52 terminals.

1.2.3.2 General Purpose – RT–11 provides several utility programs that help you perform maintenance on your system and aid in program development.

The peripheral interchange program (PIP, described in Chapter 7) is the RT–11 file maintenance program. It transfers files between the devices that are part of the RT–11 system, and it deletes and renames files as well.

The device utility program (DUP, described in Chapter 8) performs general device tasks such as initializing devices, scanning for bad blocks, duplicating device contents, and reorganizing files on the device. It operates only on RT–11 file-structured devices.

The directory listing program (DIR, described in Chapter 9) performs a wide range of directory listing operations and can list details about certain files, such as file names, file types, and block sizes.

The linker program (LINK, described in Chapter 11) converts a collection of object modules from compiled or assembled programs and subroutines into a memory image file that RT–11 can load and execute. The linker also allows you to:

- Search library files for subroutines that you specify
- Produce a load map that lists the assigned absolute addresses
- Set up a disk or memory resident overlay structure for large programs
- Create a symbol table file that lists all the global symbols used in the program
- Produce files suitable for execution as foreground jobs

The librarian program (LIBR, described in Chapter 12) lets you create and maintain libraries of functions and routines. These routines can be stored on a random-access device in library files where the linker can reference them and add them to a program's memory image file. You can create object libraries and macro libraries. The latter are used by the MACRO assembler.

The dump program (DUMP, described in Chapter 13) prints all or any part of a file or volume in octal words, octal bytes, ASCII characters, or Radix–50 characters.

The file exchange program (FILEX, described in Chapter 14) transfers files among DECsystem–10, PDP–11 RSTS/E, and DOS BATCH systems on DECtape and disks, and between RT–11 and IBM systems on diskettes.

The source file comparison program (SRCCOM, described in Chapter 15) performs a character-by-character comparison of two ASCII text files. You can request that the differences be listed in an output file or directly on the line printer or terminal to make sure that edits to a file have been performed correctly. SRCCOM can also produce a file that is suitable as input to SLP, the source file patching utility.

The binary file comparison program (BINCOM, described in Chapter 16) compares two binary files and lists the differences between them. It can provide a quick way of telling whether two data files, or output from two versions of a program, are identical. BINCOM can also produce a file that can be run as an indirect command file for the save image patch program (SIPP) to patch one file in the binary comparison so it matches the other.

The resource program (RESORC, described in Chapter 17) lists information about your system configuration and system generation special features.

The volume formatting program (FORMAT, described in Chapter 18) provides a way to format RK05, RK06, and RK07 disks, and diskettes. It also provides disk verification by writing patterns and reading them on each block of your volume.

1.2.3.3 System Jobs – RT–11 provides two utilities that you can run as system jobs if you have enabled system job support through the system generation process: Error Logging and the Queue Package. Both utilities run under the FB and XM monitors.

The Error Logger (described in Chapter 19) keeps a statistical record of all I/O transfers for each device it supports. The Error Logger also records memory parity and cache errors as they occur. With the Error Logger enabled on your system volume, you can collect data on each I/O and memory error that occurs. The Error Logger consists of three programs and a work file. This utility is a special feature; that is, you must enable it through the system generation process.

The Queue Package (described in Chapter 20) sends files to any valid RT–11 device. The Queue Package is particularly useful for queuing files for subsequent printing, although output is not restricted to the line printer. Unlike the Error Logger, the Queue Package is not a special feature. (You need not perform system generation to enable it.)

1.2.3.4 Debugging and Patching – These utility programs help you in find, diagnose, and correct programming errors.

The on-line debugging technique (ODT, described in Chapter 21) is an object module that you link with your program. It helps you debug assembled and linked programs. ODT can:

- Print and change the contents of specified locations
- Execute all or part of the object program
- Search the object program for specific bit patterns

The save image patch program (SIPP, described in Chapter 22) can patch programs that were linked with the RT–11 V04 linker. It can also patch nonoverlaid programs from versions V03 and V03B of RT–11. The major advantage in using SIPP rather than PATCH (described below) is that SIPP's format makes it easier to use.

The object module patch program (PAT, described in Chapter 23) performs minor modifications to files in object format (output files produced by the FORTRAN compiler or the MACRO assembler). It can merge several object files into one.

The source language patch program (SLP, described in Chapter 24) provides an easy way to make changes to source files. SLP can use an indirect command file (or the keyboard monitor command DIFFERENCES/SLP/OUTPUT:filespec,) created by the SRCCOM /P option to make two source files match.

The patch program (PATCH, described in Chapter 25 performs minor modifications to memory image files that are output by the pre-V04 linkers. Do not use PATCH for files linked with the V04 linker.

1.2.3.5 BATCH – The batch program (BATCH, described in Appendix A) is a complete job-control language that allows RT–11 to operate unattended.

1.2.4 Language Processors

RT–11 supports a number of language processors to help you develop programs. The *Introduction to RT–11* contains detailed information on the differences between assembly language and high-level languages. It also offers guidelines for choosing a programming language and provides demonstrations of MACRO, BASIC, and FORTRAN programs.

The MACRO-11 assembler (see Chapter 10) is part of the RT–11 system. Because MACRO-11 is an assembly language, it gives you control over the system at the most elementary level. On the other hand, it is probably more difficult to learn and use than any of the high-level languages.

The other languages RT–11 supports are:

- APL
- BASIC
- DIBOL
- FORTRAN IV

1.3 RT–11 Software Documentation

The software documentation for an RT–11 system consists of the manuals that document the RT–11 system itself, plus the documentation for any optional languages or application packages you may have.

The *RT–11 Documentation Directory* summarizes the manuals in the RT–11 documentation set. Reading this directory gives you a general picture of the topics covered in the manuals.

To find more specific information, refer to the *RT–11 Master Index*. This is a compilation of the indexes of the other RT–11 manuals. It pinpoints references by manual name and page number. It also indicates which reference is the primary source of information on the specific topic.

1.4 Obtaining System Services

The RT–11 system provides many services that allow you, for example, to copy and delete files, to examine locations in memory, to run programs, and to open and close files. Some of these services are available to you at the console terminal; others are available to application programs.

1.4.1 Using the Keyboard Monitor Commands

The keyboard monitor commands are a set of English-language commands that permit you to perform common system operations. When you type a keyboard monitor command at the console terminal, RT–11 responds by performing the operation you specify. The monitor then prompts you for another command and waits for you to respond. Chapter 4 describes the syntax and function of each of the keyboard monitor commands.

The set of keyboard monitor commands consists of two types of command: simple and complex. Simple, or direct, commands are executed directly by the keyboard monitor, and no other software components are required. The complete set of simple commands is as follows:

ASSIGN	Deposit	INSTALL	RESET	START
Base	Examine	LOAD	RESUME	SRUN
CLOSE	FRUN	R	RUN	SUSPEND
DATE	GET	REENTER	SAVE	TIME
DEASSIGN	GT	REMOVE	SET	UNLOAD

Complex, or expanded, commands are not executed directly by the keyboard monitor. Instead, a utility program or language processor is called by the keyboard monitor to perform the operations. The keyboard monitor expands the command line piece by piece and translates the command into an R command followed by a program name and one or more lines of file specifications and options for that program. When the operation completes, control returns to the keyboard monitor and it prompts you for another command. The set of complex commands is as follows:

APL	DELETE	EXECUTE	LIBRARY	SQUEEZE
BASIC	DIBOL	FOCAL	LINK	TYPE
BOOT	DIFFERENCES	FORMAT	MACRO	
COMPILE	DIRECTORY	FORTRAN	PRINT	
COPY	DUMP	HELP	RENAME	
CREATE	EDIT	INITIALIZE	SHOW	

1.4.2 Using Programs Directly

Another way to obtain services from RT–11 is to invoke system utility programs or language processors yourself, instead of invoking them indirectly through the keyboard monitor commands. By using this method you can obtain all the services provided by the complex keyboard monitor commands. (The only way to obtain the services provided by the simple keyboard monitor commands is to issue those commands.) A limited number of utility program operations are not implemented through the keyboard monitor. In addition, you must run some of the utility programs directly in order to use them at all. Programs in this group include the patching and debugging utilities.

To invoke a system utility program or a language processor run the appropriate program and specify a combination of file specifications and single alphabetic character options. Chapter 6 describes the syntax you use to interact with the utility programs and language processors. Chapters 7 through 20 contain detailed information on each program.

1.4.3 The Relationship Between Complex Commands and System Programs

It is possible to obtain the services provided by the complex keyboard monitor commands by directly running the appropriate system programs. Appendix B provides a complete list of the keyboard monitor commands and the system programs they invoke.

The following example demonstrates two ways of copying a listing of a program from the system disk, where it is stored as MYFILE.LST, to the line printer. The keyboard monitor command to do this is as follows:

```
.PRINT MYFILE<RET>
```

The commands to invoke a utility program, specify the same operation, and return control to the monitor are:

```
.R PIP<RET>
*LP:=DK0:MYFILE.LST<RET>
*^C
```

So, although there are two ways of obtaining the same services, bear in mind that the syntax for using the utility programs and language processors is quite different from the keyboard monitor command syntax. Since the

keyboard commands are designed to be easy to remember and easy to use, it makes sense to use them whenever possible.

1.4.4 The System Macro Library and Programmed Requests

The system macro library, called SYSMAC.SML, contains routines that you can use in MACRO assembly language programs and in device handlers. You call the routines in your assembly language program, and they expand into lines of source code. These macros can save you considerable programming effort. See the *RT–11 Programmer's Reference Manual.*

1.4.5 SYSLIB FORTRAN – Callable Subprograms

Most of the system subroutine library (SYSLIB) routines are written in MACRO. They give the FORTRAN programmer many of the services that the MACRO programmer can obtain from the system macro library (SYSMAC.SML). These subprograms can be called from a program written in any programming language, as long as the program conforms to the FORTRAN calling conventions described in the *RT–11 Programmer's Reference Manual.*

Chapter 2
Program Development

The number and type of tools available on any system depend on many factors, including the size of the system, its application, and its cost. RT–11 provides several program development aids, including an editor, an assembler, a linker, a debugger, and a librarian. High-level languages, such as FORTRAN or BASIC, are optionally available.

This chapter describes briefly the program development cycle, which is illustrated in Figure 2–1. The *Introduction to RT–11* contains a much more thorough treatment of program development including demonstrations of MACRO, BASIC, and FORTRAN programs.

2.1 Using an Editor

You use an editor to create and modify textual material. Text may be the statements in a source program, or any other ASCII data, such as reports or memos. In this respect, using an editor is analogous to using a typewriter; you sit at a keyboard and type text. However, the functions of an editor far exceed those of a typewriter. Once a text file has been created, you can modify, relocate, replace, merge, or delete text, all by means of simple editing commands. When you are satisfied with your text, you can save it on a storage device where it is available for later reference.

2.2 Using the Assembler

Program development does not stop with the creation of a source program. Since the computer cannot understand any language but machine language, you need an intermediary program to convert source code into the instructions the computer can execute. This is the function of an assembler.

The assembler accepts alphanumeric representations of PDP–11 coding instructions, interprets the code, and produces as output the appropriate machine, or object, code. You can direct the assembler to generate a listing of both the source code and binary output, as well as more specific listings that are helpful during the program debugging process. In addition, the assembler is capable of detecting certain common coding errors and issuing appropriate warnings.

The assembler's output is called object output because it is composed of object, or binary, code. On PDP–11 systems, the object output is called a module; it contains your source program in the binary language that, when linked, is acceptable to a PDP–11 computer.

2.3 Using the Linker

Source programs may be complete and functional by themselves; however, some programs are written in such a way that they must be used with other programs or modules to form a complete and logical flow of instructions. For this reason, the object code produced by the assembler must be relocatable. That is, assignment of memory locations must be deferred until the code is combined with all other necessary object modules. The linker performs this function.

The linker combines and relocates separately assembled object programs. The output produced by the linker is a load module, the final linked program that is ready for execution. You can, if you wish, request a load map that displays all addresses assigned by the linker.

2.4 Using the Debugger

You can rarely create a program that does not contain at least one error, either in the logic of the program or in its coding. You may discover errors while you are editing the program, or the assembler may find errors during the assembly process and inform you by means of error codes. The linker may also catch certain errors and issue appropriate messages. Often, however, it is not until execution that you discover that your program is not working properly. Programming errors may be extremely difficult to find, and for this reason, a debugging tool, ODT (described in Chapter 21), is available to help you find the cause of errors.

ODT allows you to control the execution of your program interactively. With it, you can examine the contents of individual locations, search for specific bit patterns, set designated stopping points during execution, change the contents of locations, continue execution, and test the results — all without editing and reassembling the program.

Note that it is advisable to test new programs by having them process data for which results are already known. If the results do not match, you know you have errors.

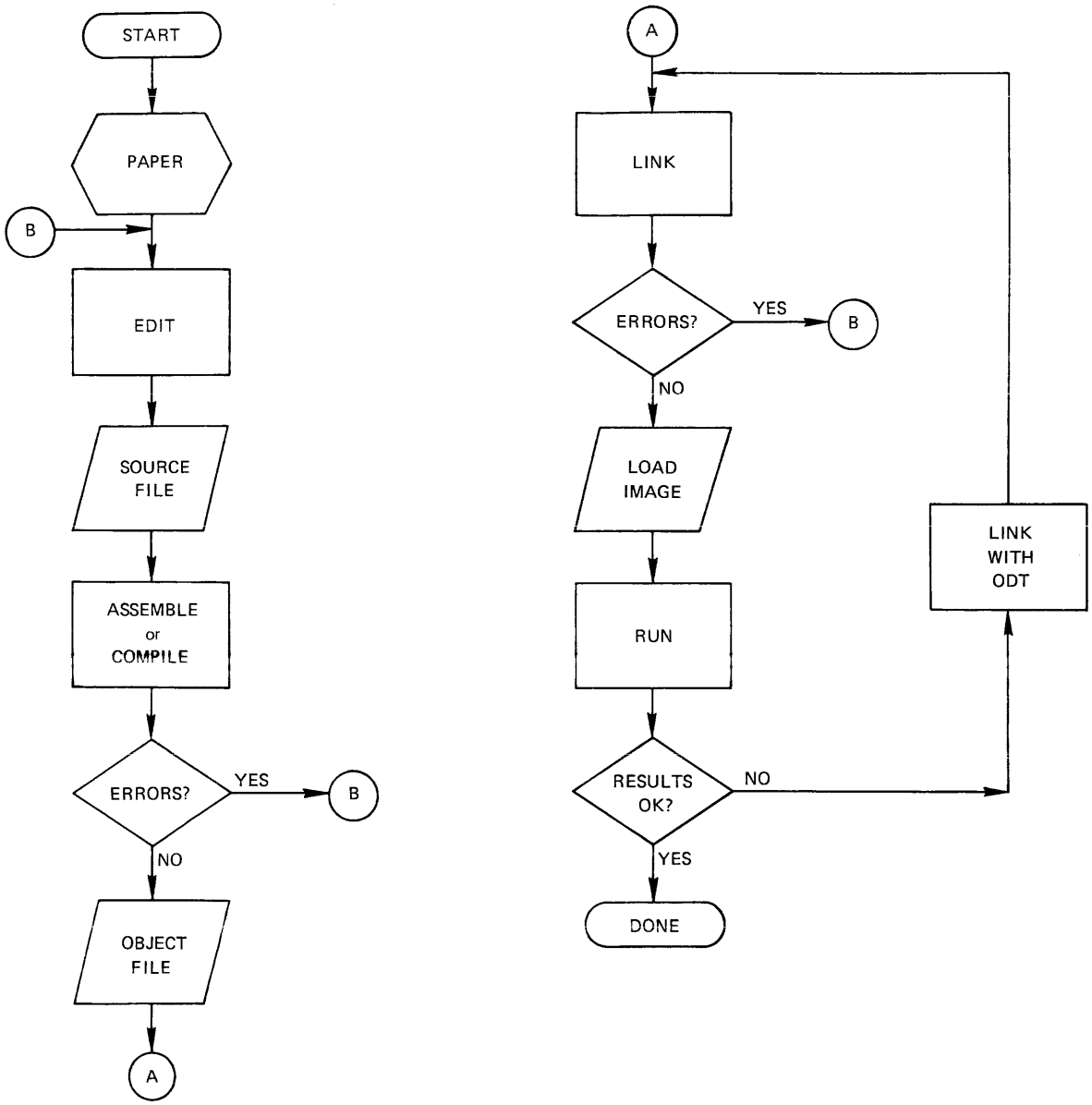
2.5 Using the Librarian

When programs are written and debugged, they are useful to other programmers. Often, routines that are common to many programs, such as input and output routines, or sections of code that are used over and over again, are more useful if they are placed in a library where they can be retrieved by any interested user. A librarian provides such a service by allowing creation of a library file. Once created, the library can be expanded or updated, or a directory of its contents can be listed.

2.6 Using a High-level Language

High-level languages simplify your work by providing an alternative means, other than assembly language, of writing a source program. Generally, high-level languages are easy to learn. A single command causes the computer to perform many machine-language instructions. You do not need to know about the mechanics of the computer to use a high-level language. In addition, some high-level languages, such as BASIC, offer a special immediate mode that allows you to solve equations and formulas as though you were using a calculator. You can concentrate on solving the problem rather than on using the system.

Figure 2–1: Program Development Cycle



Part II
System Communication

The monitor is the center of RT–11 system communications; it provides access to system and user programs, performs input and output functions, and controls foreground and background jobs.

You communicate with the monitor through keyboard commands and programmed requests. You can use the keyboard commands (described in Chapter 4) to load and run programs, start or restart programs at specific addresses, modify the contents of memory, and assign and deassign alternate device names, to name only a few of the functions.

Programmed requests (described in detail in the *RT–11 Programmer's Reference Manual*) are source program instructions that request the monitor to perform monitor services. These instructions allow assembly language programs to use the monitor features. A running program communicates with the monitor through programmed requests. FORTRAN programs have access to programmed requests through the system subroutine library (SYSLIB). Programmed requests can, for example, manipulate files, perform input and output, and suspend and resume program operations.

Of the two chapters in this part, Chapter 3 describes system conventions and contains information that helps you get started with RT–11; Chapter 4 introduces the keyboard monitor commands, which are your means of controlling the RT–11 system.

Chapter 3
System Conventions

This chapter contains information that will help you start using the RT–11 system. It describes:

- Startup procedure
- Data formats
- Physical device names
- File names and file types
- Device structures
- Special function keys
- Foreground/background terminal I/O
- Type-ahead feature

Before you operate the RT–11 system, you should be familiar with the special character commands, file naming procedures and other conventions that are standard for the system. These conventions are described in this chapter.

3.1 Startup Procedure

For information on building the system and loading the monitor, refer to the *Introduction to RT–11*, to the *RT–11 Installation and System Generation Guide*, or to any instructions provided by your DIGITAL representative.

When the system has been built and you load the monitor into memory, the monitor prints one of the following identification messages on the terminal:

RT–11SJ (S) Vxx.nnp

RT–11FB (S) Vxx.nnp

RT–11XM (S) Vxx.nnp

The message indicates which monitor (SJ, FB, or XM) is loaded; you establish which is to be loaded when you install the system. The (S) indicates that the monitor was created through the system generation process, if applicable.

Vxx represents the version and release number of the monitor — for example, V04, for Version 4 (release A). *nnp* represents the library submission number and the patch level — for example, 01B, for library number 1 (patch level B).

As soon as a monitor takes control of the system, it attempts to execute keyboard monitor commands from a startup indirect command file called STARTS.COM, for the SJ monitor; STARTF.COM, for the FB monitor; or STARTX.COM for the XM monitor. You can place commands in this startup file that will perform routine tasks such as assigning logical device names to physical devices, or setting the current date. If the monitor does not find the appropriate file, it issues a warning message. After executing the startup indirect command file, the system prints its prompt (.) indicating that it is ready to accept commands. You should now write-enable the system device. (Note that if you do not want the startup indirect command file feature, you can disable it during system generation or you can apply a customization patch.)

3.2 Data Formats

The RT–11 system stores data in two formats: ASCII and binary. The binary data can be organized in many formats, including object, memory image, relocatable image, and load image.

Files in ASCII format conform to the American Standard Code for Information Interchange, in which each character is represented by a 7-bit code. Files in ASCII format include program source files created by the editor and BASIC, listing and map files created by various system programs, and data files consisting of alphanumeric characters.

Files in binary object format consist of data and PDP–11 machine language code. Object files are the files the assembler or language compiler produces; they are used as input to the linker.

The linker can produce runnable files in one of three formats: (1) memory image format (.SAV), (2) relocatable image format (.REL), or (3) load image format (.LDA).

A memory image file (.SAV) is a picture of what memory looks like after you load a program. The file itself requires the same number of disk blocks as the corresponding number of 256-word memory blocks. A memory image file does not require relocation and can run in an SJ environment, as a background program under the FB or XM monitor, or as a foreground virtual job under the XM monitor.

A relocatable image file (.REL) is linked as though its bottom address were 1000, but relocation information is included with its memory image. When you call the program with the FRUN command, the file is relocated as it is loaded into memory. A relocatable image file can run in a foreground environment.

You can produce a load image (.LDA) file for compatibility with the PDP–11 paper tape system. The absolute binary loader loads this file. You can load and execute load image files in stand-alone environments without relocating them.

3.3 Physical Device Names

When you request services from the monitor, you must sometimes specify a peripheral device. You can specify devices by means of a standard two-character physical device name. Table 3–1 lists each name and its related device. If you do not specify a unit number for devices with more than one unit, the system assumes unit 0.

Table 3–1: Permanent Device Names

Permanent Name	I/O Device
CR:	CR11/CM11 Card Reader
CTn:	TA11 Cassette (*n* is 0 or 1)
DDn:	TU58 DECtape II (*n* is an integer in the range 0–3)
DK:	The default logical storage device for all files (DK: is initially the same as SY:)
DKn:	The specified unit of the same device type as the system device
DLn:	RL01, RL02 Disk (*n* is an integer in the range 0–3)
DMn:	RK06, RK07 Disk (*n* is an integer in the range 0–7)
DPn:	RP02, RP03 Disk (*n* is an integer in the range 0–7)
DSn:	RJS03/4 Fixed-Head Disks (*n* is an integer in the range 0–7)
DTn:	DECtape (*n* is an integer in the range 0–7)
DXn:	RX01 Diskette (*n* is an integer in the range 0–3)
DYn:	RX02 Diskette (*n* is an integer in the range 0–3)
LP:	Line Printer
LS:	Serial Line Printer (a hard copy output device connected to a DL11 interface)
MMn:	TJU16/TU45 (industry-compatible) Magtape (*n* is an integer in the range 0–7)
MQ:	Message queue pseudo-device for inter-job communication under the FB monitor.
MSn:	TS11 Magtape (*n* is an integer in the range 0–7)
MTn:	TM11/TMA11/TS03/TE16 (industry-compatible) Magtape (*n* is an integer in the range 0–7)
NL:	Null device
PC:	PC11 Combined High-Speed Paper Tape Reader and Punch
PDn:	The mass storage volume for the PDT–130/150 intelligent terminal. Volumes are DECtape II or single-density diskettes (*n* is either 0 or 1).
RF:	RF11 Fixed-Head Disk Drive
RKn:	RK05 Disk Cartridge Drive (*n* is an integer in the range 0–7)
SY:	The default logical system device; the device and unit from which the system is bootstrapped
SYn:	The specified unit of the same device type as SY:
TT:	Console Terminal Keyboard and Printer

In addition to using the permanent names shown in Table 3–1, you can assign logical names to devices. A logical name takes precedence over a physical name and thus provides device independence. With this feature, you do not have to rewrite a program that is coded to use a specific device if the device becomes unavailable. You associate logical names with physical devices by using the ASSIGN command, which is described in Section 4.4.

3.4 File Names and File Types

You can reference files symbolically by a name of one to six alphanumeric characters (followed, optionally, by a period and a file type of up to three alphanumeric characters). No spaces or tabs are allowed in the file name or file type. The file type generally indicates the format or contents of a file. It is good practice to conform to the standard file types for RT–11. If you do not specify a file type for an input or output file, most system programs use or assign an appropriate default file type. Table 3–2 lists the standard file types used in RT–11.

Table 3–2: Standard File Types

File Type	Meaning
.BAD	Files with bad (unreadable) blocks; you can assign this file type whenever bad areas occur on a device. The .BAD file type makes the file permanent in that area, preventing other files from using it and consequently becoming unreadable
.BAK	Editor backup file
.BAS	BASIC source file (BASIC input)
.BAT	BATCH command file
.CND	System generation conditional file
.COM	Indirect command file
.CTL	BATCH control file generated by the BATCH compiler
.CTT	BATCH internal temporary file
.DAT	BASIC or FORTRAN data file
.DBL	DIBOL source file
.DDF	DIBOL data file
.DIF	SRCCOM output file
.DIR	Directory listing file
.DMP	DUMP output file
.FOR	FORTRAN IV source file (FORTRAN input)
.LDA	Absolute binary (load image) file (optional linker output)
.LOG	BATCH log file
.LST	Listing file (MACRO, FORTRAN, LIBR, or DIBOL output)
.MAC	MACRO source file (MACRO or SRCCOM input, LIBR input and output)
.MAP	Map file (linker output)

(continued on next page)

Table 3–2: Standard File Types (Cont.)

File Type	Meaning
.OBJ	Relocatable binary file (MACRO or FORTRAN output, linker input, LIBR input and output)
.REL	Foreground job relocatable image (linker output, default for monitor FRUN command)
.SAV	Memory image; default for R, RUN, SAVE, and GET keyboard monitor commands; also default for linker output
.SML	System MACRO library
.SLP	SLP command file
.SOU	Temporary source file generated by BATCH
.STB	Symbol table file in object format containing all the symbols produced during a link
.SYS	Monitor files, handlers, and system job files
.TMP	ERROUT temporary file; QUEUE work file
.TXT	Text file

3.5 Device Structures

RT–11 devices are categorized according to two characteristics: (1) the device's method of processing information, and (2) the device's physical structure.

All RT–11 devices are either randomly accessed or sequentially accessed. Random-access devices allow the system to process blocks of data in a random order — that is, independent of the data's physical location on the device or its location relative to any other information. All disks, diskettes, DECtape, and DECtape II fall into this category. Random-access devices are sometimes called block-replaceable devices, because you can manipulate (rewrite) individual data blocks without affecting other data blocks on the device.

Sequential-access devices require sequential processing of data; the order in which the system processes the data must be the same as the physical order of the data. RT–11 devices that are sequential devices are cassette, paper tape reader and punch, card reader, line printer, terminal, and the null device.

File-structured devices are those devices that allow the system to store data under assigned file names. RT–11 devices that are file-structured include all disk, diskette, DECtape, DECtape II, magtape, and cassette devices. Non-file-structured devices, however, do not store files; they contain a single logical collection of data. These devices, which include the line printer, card reader, terminal, and paper tape reader and punch, are generally used for reading and listing information.

File-structured devices that have a standard RT–11 directory at the beginning are RT–11 directory-structured devices. A device directory consists of a series of directory segments that contain the names and lengths of the files on that device. The system updates the directory each time a program moves, adds, or deletes a file on the device. (The *RT–11 Software Support Manual* contains a more detailed explanation of a device directory.) RT–11 directory-structured devices include all disks and DECtapes. Some devices that do not have the standard RT–11 directory structure, such as magtape and cassette, store directory information at the beginning of each file, but the system must read the device sequentially to obtain all information about all files.

Table 3–3 shows the relationships among devices, access methods, and structures.

Table 3–3: Device Structures

Device	Structure			
	File	Non-file	RT-directory	Non-RT-directory
Random Access				
Disk, diskette	x		x	
DECtape, DECtape II	x		x	
Sequential Access				
Magtape	x			x
Cassette	x			x
Paper tape		x		
Card reader		x		
Line printer		x		
Terminal		x		

3.6 Special Function Keys

Special function keys and keyboard commands let you communicate with the RT–11 monitor to allocate system resources, manipulate memory images, start programs, and use foreground/background services.

The special functions of certain terminal keys you need for communication with the keyboard monitor are explained in Table 3–4. In the FB system, the keyboard monitor runs as a background job when no other background job is running.

Enter CTRL commands by holding the CTRL key down while typing the appropriate letter.

Table 3–4: Special Function Keys

Key	Function
CTRL/A	Is valid only after you type the monitor GT ON command and use the display. CTRL/A, a command that does not echo on the terminal, pages output if you use it after a CTRL/S. The system permits console output to resume until the screen is completely filled; text currently displayed scrolls upward off the screen. CTRL/A has no special meaning if the keyboard monitor command GT ON is not in effect.
CTRL/B	Causes the system to direct all keyboard input to the background job. The FB monitor echoes B> on the terminal. The system takes at least one line of output from the background job. The foreground or system job, however, has priority, so the system returns control to the foreground or system job when it has output. In multi-terminal systems, CTRL/B has no special meaning if the background console is not shared. CTRL/B directs all typed input to the background job until a CTRL/F redirects input to the foreground job or a CTRL/X directs input to a system job. CTRL/B has no special meaning when used under a single-job monitor or when a SET TT NOFB command is in effect.
CTRL/C	Terminates program execution and returns control to the keyboard monitor. CTRL/C echoes ^C on the terminal. You must type two CTRL/Cs to terminate execution unless the program to be terminated is waiting for terminal input or is using the TT handler for input. In these cases, one CTRL/C terminates execution. Under the FB monitor, the job that is currently receiving input is the job that is stopped (determined by the most recently typed command, CTRL/F or CTRL/B). To make sure that the command is directed to the proper job, type CTRL/B or CTRL/F before typing CTRL/C.
CTRL/E	Causes all terminal output to appear on both the display screen and the console terminal simultaneously. CTRL/E is valid after you type the monitor GT ON command and use the display. The command does not echo on the terminal. A second CTRL/E disables console terminal output. CTRL/E has no special meaning if GT ON is not in effect.
CTRL/F	Causes the system to direct all keyboard input to the foreground job and take all output from the foreground job. The FB monitor echoes F> on the terminal unless output is already coming from the foreground job. If no foreground job exists, the monitor prints an error message (*F?*). Otherwise, control remains with the foreground job until redirected to the background job (with CTRL/B), or redirected to a system job (with CTRL/X), or until the foreground job terminates. In multi-terminal systems, CTRL/F has no special meaning if the foreground console is not shared. CTRL/F has no special meaning when used under a single-job monitor, or when a SET TT NOFB command is in effect.
CTRL/O	Causes RT–11 to suppress terminal output while continuing program execution. CTRL/O echoes as ^O on the terminal. RT–11 reenables terminal output when one of the following occurs: 1. You type a second CTRL/O. 2. You return control to the monitor by typing CTRL/C or by issuing the .EXIT request in your program. 3. The running program issues a .RCTRLO programmed request (see the *RT–11 Programmer's Reference Manual*). RT–11 system programs reset CTRL/O to the echoing state each time you enter a new command string. Note that when you are using CTRL/O under the single-job monitor, the system can print an extraneous character after the monitor echoes the CTRL/O and a carriage return/line feed.

(continued on next page)

Table 3–4: Special Function Keys (Cont.)

Key	Function
CTRL/Q	Resumes printing characters on the terminal from the point printing previously stopped because of a CTRL/S. CTRL/Q does not echo and has no special meaning under a multi-terminal SJ or FB monitor if a SET TT NOPAGE command is in effect.
CTRL/S	Temporarily suspends output to the terminal until you type a CTRL/Q. CTRL/S does not echo. Under a multi-terminal SJ or FB monitor, CTRL/S is not intercepted by the monitor if TT NOPAGE is in effect.
CTRL/U	Deletes the current input line and echoes as ^U followed by a carriage return at the terminal. (The current line is defined as all characters back to, but not including, the most recent line feed, CTRL/C, or CTRL/Z.)
CTRL/X	Causes the system to prompt you for a job name, then to direct all keyboard input to the system job you specify. When you type CTRL/X, the system prints *Job?* at the terminal. Specify the system job name (or logical job name) of the system job to which you want to direct input. Specify B or F to direct keyboard input to the background or foreground job, respectively. If the specified job does not exist, the system prints a question mark (*?*), otherwise it prints the system job name at the terminal. Control remains with the specified system job until the job terminates, or control is redirected to the background job (with CTRL/B), the foreground job (with CTRL/F), or another system job (with CTRL/X). CTRL/X has no special meaning when used with a monitor that does not have system job support or when SET TT NOFB.
CTRL/Z	Terminates input when used with the terminal device handler (TT). It echoes as ^Z on the terminal. The CTRL/Z itself does not appear in the input buffer. Note that because CTRL/Z is a line terminator, you cannot delete it, once typed. If TT is not being used, CTRL/Z has no special meaning.
DELETE or RUBOUT	Deletes the last character from the current line and echoes a backslash plus the character deleted. Each RUBOUT succeeding DELETE deletes and echoes another character. The system prints an enclosing backslash when you type a key other than DELETE. This erasure is performed from right to left up to the beginning of the current line. If you are using a video display terminal and you have issued the SET TT SCOPE command, DELETE erases characters with a backspace, space, backspace sequence. Your corrections appear on the screen; RUBOUT does not enclose them with backslash characters.

3.7 Foreground/Background Terminal I/O

Console input and output under FB are independent functions; therefore, you can type input to one job while another job prints output. You may be in the process of typing input to one job when the system is ready to print output from another job on the terminal. In this case, the job that is ready to print interrupts you and prints the message on the terminal; the system does not redirect input control to this job, however, unless you type a CTRL/B, CTRL/F, or CTRL/X, whichever applies. If you type input to one job while another has output control, the system suppresses the echo of the input until the job accepting input gains output control; at this point, all accumulated input echoes.

If the two jobs are ready to print output at the same time, the job with the higher job number has priority. For example, in an FB system, the system prints output from the foreground job until it encounters a line feed. Each time the system prints a line feed, it checks to see if the foreground job (or, in a monitor with system job support, any higher priority job) has output; if so, the system gives control to the highest priority job that is ready to print.

When the foreground job terminates, control reverts automatically to the background job.

3.8 Type-ahead Feature

The monitor has a type-ahead feature that lets you enter terminal input while a program is executing. For example:

```
.DIRECTORY/PRINTER
DATE
```

While the first command line is executing, you can type the second line. Although the system echoes the characters you type immediately after you type them, the system stores this terminal input in a buffer and uses it when the system completes the first operation.

If type-ahead input exceeds the input buffer capacity (usually 134 characters), the terminal bell rings and the system accepts no characters until a program uses part of the type-ahead buffer, or until you delete characters. No input is lost. Type-ahead is particularly useful when you specify multiple command lines to system programs.

Note that after you bootstrap any RT–11 monitor, the system does not recognize the type-ahead feature until either the keyboard prompting character (.) prints or the startup indirect command file begins executing. If you type ahead prior to this, the system either ignores or truncates your type-ahead.

If you type a single CTRL/C while the system is in this mode, the system puts CTRL/C into the buffer. The program currently executing exits when it makes a terminal input request. Typing a double CTRL/C returns control to the monitor immediately. If you terminate a job by typing two CTRL/Cs, the system discards any unprocessed type-ahead.

Chapter 4
Keyboard Commands

Keyboard commands allow you to communicate with the RT–11 system. You enter keyboard commands at the terminal in response to the keyboard monitor dot (.), and the operating system invokes the appropriate system programs to service these commands.

This chapter uses some symbolic conventions to describe the monitor command language. The preface to this manual contains a more detailed list of the symbolic conventions used throughout the manual. You should familiarize yourself with the symbols and their meaning before reading this chapter.

4.1 Command Syntax

The system accepts commands as either: (1) a complete string containing all the information necessary to execute a command, or (2) as a partial string. In the latter case the system prompts you to supply the rest of the information. Terminate each command with a carriage return.

The general syntax for a command is:

```
command[/option...] input-filespec[/option...]
output-filespec[/option...]

or

command[/option...]
prompt1? input-filespec[/option...]
prompt2? output-filespec[/option...]
```

where:

command	is the command name
/option	represents a command qualifier that specifies the exact action to be taken. Any option you supply immediately following the command applies to the entire command string
prompt	represents the keyboard monitor prompt for more information. The keyboard monitor prints an appropriate prompt only if you omit input and/or output file or device specifications in the initial command line (Note that not all keyboard monitor commands print prompts.)
input-filespec	represents the file on which the action is to be taken

/option	represents a file qualifier that specifies more detailed information about that particular file or action to be taken
output-filespec	represents the file that is to receive the results of the operation
/option	represents a file qualifier that specifies more detailed information about that particular file or action to be taken

In the alphabetical listing of keyboard monitor commands in Section 4.4, each command begins with a graphic presentation of the syntax involved (see Figure 4–1 for an illustration of a typical command). These presentations provide a ready-reference list of the options that the commands accept, as well as information that makes the commands easier to use. The following list describes the conventions used.

1. Capital letters represent command names or options, which you must type as shown. (Abbreviations are discussed later in this section.)

2. Lower-case letters represent arguments or variables, for which you must supply values. For options that accept numeric arguments, the system interprets the values as decimal, unless otherwise stated. Some values, usually memory addresses, are interpreted as octal; these cases are noted in the accompanying text.

3. Square brackets ([]) enclose options; you can include the item that is enclosed in the brackets or you can omit it, as you choose. If a vertical list of items is enclosed in square brackets, you can combine the options that appear in the list. However, an option set off from the others by blank lines (see /BOOT and /DEVICE in Figure 4–1) indicates that you cannot combine that option with any other option in the list.

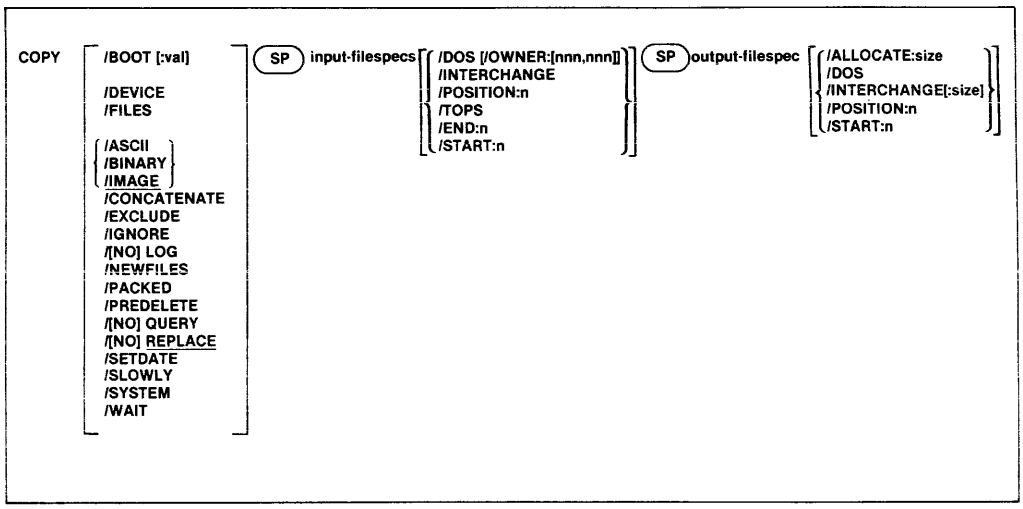
4. Braces ({ }) enclose options that are mutually exclusive. You can choose only one option from a group of options that appear in braces.

5. It is conventional to place command options (those qualifiers that apply to the entire command line) immediately after the command. However, it is also acceptable to specify a command option after a file specification. File options (those that qualify a particular file specification) must appear in the command line directly after the file to which they apply. The graphic representation of each command shows which options are file qualifiers, and whether they must follow input or output file specifications.

6. A line such as [NO]QUERY represents two mutually exclusive options: QUERY and NOQUERY.

7. Underlining indicates default options, that is, the option that the system uses if you do not specify any choice of action.

Figure 4–1: Sample Command Syntax Illustration



A filespec represents a specific file and the device on which it is stored. Its syntax is:

dev:filnam.typ

where:

dev: represents either a logical device name or a physical device name, which is a two- or three-character name from Table 3–1

filnam represents the one- to six-character alphanumeric name of the file

.typ represents the one- to three-character alphanumeric file type, some of which are listed in Table 3–2

There are several ways to indicate the device on which a file is stored. You can explicitly type the device name in the file specification:

```
DX1:TEST.LST
```

You can omit the device name:

```
TEST.LST
```

In this case, the system assumes that the file is stored on device DK:.

4.1.1 Factoring File Specifications

If you want to specify several files on the same device, you can use factoring. That is, you can enclose multiple file names in parentheses, as in the following example:

```
DT0:(TEST.LST,TESTA.LST,TESTB.LST)
```

The command shown above has the same meaning as and is easier to use than the next command:

```
DT0:TEST.LST,DT0:TESTA.LST,DT0:TESTB.LST
```

When you use factoring the device name outside the parentheses applies to each file specification inside the parentheses. Without factoring, the system interprets each file specification to be *DK:filespec* unless you explicitly specify another device name.

Factoring is useful for complicated command lines. It is a general method of string replacement that you can use in many different situations. The monitor uses the following algorithm to interpret command lines that require factoring:

Format of the command line you type:

D1 T1 (T3 D3 T4 D4...Tn) T2 D2

Format of the command line after the monitor performs the factoring:

D1 T1T3T2 D3 T1T4T2 D4...T1TnT2 D2

In the skeleton examples shown above, the symbols have the following meaning:

D represents a delimiter

D1 is one of the following delimiters:

comma
space
beginning of line

D2 is one of the following delimiters:

comma
space
slash
end of line

D3 through Dn can be one of the following delimiters:

comma
space

T represents a text string

The following example shows how a command line expands after factoring. Note that the /SYSTEM option appears only once in the resulting output line.

Original command line:

```
COPY DX:FIL(1,2,3).SYS/SYSTEM RK1:
```

Resulting command line (after factoring):

```
COPY DX:FIL1.SYS,DX:FIL2.SYS,DX:FIL3.SYS/SYSTEM RK1:
```

NOTE

There is a restriction on the use of factoring. The command string that results from the expansion of the line you enter must not exceed 80 characters in length. If you use six-character file names and you also use factoring, specify only five files in a command line.

4.1.2 File Type Specification

If you omit the file type in a file specification, the system assumes one of a number of defaults, depending on which command you issue. The MACRO command, for example, assumes a file type of .MAC for the input file specification, and the PRINT command assumes .LST. Some commands (such as COPY) do not assume a particular file type, and may assume a wildcard default (see Section 4.2). If you need to specify a file that has no file type in a command that assumes a default file type, type a period after the file name. For example, to run the file called TEST, type:

```
RUN TEST.
```

If you omit the period after the file name, the system assumes a .SAV file type and tries to execute a file called TEST.SAV.

You can enter up to six input files and up to three output files for some commands. If the command string does not fit on one line of your terminal, use the hyphen (-) continuation character as the last character in the line to break the string into smaller sections. Use a carriage return to terminate the command string. Note that there is still a limit of 80 characters for an expanded command line.

Some of the command and file options are mutually exclusive. You should avoid using combinations of options that give contradictory instructions to the system. For example,

```
.DELETE/QUERY/NOQUERY TEST.LST
```

This command is not meaningful. Some mutually exclusive options are less obvious; these are noted, where necessary, in the list of options following each command and are enclosed by braces ({ }) in the graphic representations of the command syntax.

4.1.3 Abbreviating Keyboard Commands

Although the keyboard monitor commands are all English-language words and therefore easy to use, it can become tedious to type words like CROSS-REFERENCE and ALLOCATE frequently. You can use as abbreviations the minimum number of characters that are needed to make the command

or option unique. Table B–1 in Appendix B lists the minimum abbreviations for the commands and options. Note also that in Section 4.4, the abbreviations are in red.

An easy way to abbreviate the commands, and one that is always correct, is to use the first four characters or the first six characters if the qualifier starts with NO. For example:

CONCATENATE can be shortened to CONC

NOCONCATENATE can be shortened to NOCONC

The system prints an error message if you use an abbreviation that is not unique. For example, typing the following command produces an error, because C could mean COPY or COMPILE.

```
C TEST.LST
```

4.1.4 Keyboard Prompts

The prompting form of the command may be easier for you to learn if you are a new user. If you type a command followed by a carriage return, the system prompts you for an input file specification:

```
COPY/CONCATENATE
From?
```

You should enter the input file specification and a carriage return:

```
DX1:(TEST.LST,TESTA.LST)
```

The system prompts you for an output file specification:

```
To ?
```

You should enter the output file specification and a carriage return:

```
DX2:TEST.LST
```

The command now executes.

The system continues to prompt for an input and output file specification until you provide them. If you respond to a prompt by entering only a carriage return, the prompt prints again. You can combine the normal form of a command with the prompting form, as this example shows:

```
.COPY    ABC.LST
To  ?    DEF.LST
```

The system always prompts you for information if any required part of the command is missing. You can also enter just an option in response to a prompt. The two following examples are equivalent.

```
.COPY
From ?    *.MAC/NOLOG
To   ?    *.BAK
```

```
.COPY
From ?    /NOLOG
From ?    *.MAC
To   ?    *.BAK
```

4.2 Wildcards

Some commands accept wildcards (% and *) in place of the file name, file type, or characters in the file name or file type. The system ignores the contents of the wild field and selects all the files that match the remaining fields.

An asterisk (*) can replace a file name:

```
*.MAC
```

The system selects all files on device DK: that have a .MAC file type, regardless of their name.

An asterisk (*) can replace a file type:

```
TEST.*
```

The system selects all files on device DK: that are named TEST, regardless of their file type.

An asterisk (*) can replace both a file name and a file type:

```
*.*
```

The system selects all files on device DK:.

An embedded asterisk (*) can replace any number of characters in the input file name or file type:

```
A*B.MAC
```

The system selects all files on device DK: with a file type of .MAC whose file names start with A and end with B. For example, AB, AXB, AXYB, etc., would be selected.

The percentage symbol (%) is always considered to be an embedded wildcard. It can replace a single character in the input file name or file type:

```
A%B.MAC
```

The system selects all files on device DK: with a file type of .MAC whose file names are three characters long, start with A, and end with B. For example, AXB, AYB, AZB, etc., would be selected.

Table 4–1 lists commands that support wildcards.

Table 4–1: Commands Supporting Wildcards

Command	Specification	
	Input File	Output File
COPY	X	X
DELETE	X	
DIRECTORY	X	
HELP	X	
PRINT	X	
RENAME	X	X
TYPE	X	

For commands that support wildcards the system has a special way of interpreting the file specifications you type. You can omit certain parts of the input and output specifications, and the system assumes an asterisk (*) for the omitted item. Table 4–2 shows the defaults that the system assumes for the input and output specifications of the valid commands.

Table 4–2: Wildcard Defaults

Command	Default	
	Input	Output
COPY, RENAME	*.*	*.*
DIRECTORY	DK:*.*	
PRINT, TYPE	*.LST	
DELETE	filnam.*	

For example, if you need to copy all the files called MYPROG from DK: to DX1:, use this command:

```
.COPY/NOQUERY MYPROG  DX1:
```

The system interprets this command to mean:

```
.COPY/NOQUERY  DK:MYPROG.*  DX1:*.*
```

The system copies all the files called MYPROG, regardless of their file type, to device DX1: and gives them the same names.

If you need a directory listing of all the files on device DK:, type the following command:

```
.DIRECTORY
```

The system interprets this command to mean:

```
.DIRECTORY  DK:*.*
```

To list on the printer all the files on device DK: that have a .LST file type, use this command:

```
.PRINT -DK:
```

The system interprets this command to mean:

```
.PRINT  DK:*.LST
```

To delete all the files on device DK: called MYPROG, regardless of their file type, use this command:

```
.DELETE/NOQUERY  MYPROG
```

The system interprets this to mean:

```
.DELETE/NOQUERY  DK:MYPROG.*
```

You can use the SET WILDCARDS EXPLICIT command (described in Section 4.4) to change the way the system interprets these commands.

4.3 Indirect Files

You can group together as a file a collection of keyboard commands that you want to execute sequentially. This collection is called an indirect command file, or indirect file. Indirect files are best suited to perform tasks that require a significant amount of computer time and that do not require your supervision or intervention. Any series of commands that you are likely to type often can also run easily as an indirect file. The indirect file concept is similar to BATCH processing. Although indirect files lack some of the capabilities of BATCH, they are easier to use, use the same commands as normal operations, and generally require less memory overhead than the BATCH processor. (RT–11 BATCH is described in Appendix A of this manual.) This section describes how to create indirect files and how to execute them.

4.3.1 Creating Indirect Files

Create an indirect file by using the EDIT/CREATE command described in Section 4.4. It is conventional to use a .COM file type for an indirect file, but you can choose any file name that you wish. Structure the lines of text to look like keyboard input, placing one command on each line of the file and terminating each line with a carriage return. Do not include the prompt character (.) in the line. Any keyboard monitor command you can type at the terminal you can also include in an indirect file. The following file, for example, prints the date and time, and creates backup copies of all FORTRAN source files:

```
DATE
TIME
COPY *.FOR *.BAK
```

Control returns to the monitor at the console terminal after this indirect file executes.

In addition to using the keyboard monitor commands, you can also run one of the RT–11 system utility programs in an indirect file. In this case, structure your input to conform to the Command String Interpreter syntax described in Chapter 6. Do not include the CSI asterisk (*) in any line that provides input or output to a utility program. The following file starts the directory system utility program and lists the directory of two devices on the line printer.

```
R DIR
LP:=CT0:/C:3
LP:=DT1:/C:3
^C
```

Note that the last command line is ^C. This is not the standard CTRL/C sequence you enter by holding down the CTRL key and typing a C. Rather, it is a readable character sequence that consists of two separate characters: a circumflex (uparrow) followed by a C. This sequence represents CTRL/C in indirect files. This sequence terminates the directory program so that control returns to the monitor when the indirect file finished executing. Otherwise, the directory program would be left waiting for input from the console terminal when the indirect file finished executing.

Remember to terminate the last command line with a carriage return, as you would any other line.

NOTE

If you have a small (12K) configuration or a very large indirect command file, use frequent CTRL/Cs in your indirect files. When the system processes an indirect file, it first places each line in a special memory buffer. This memory buffer must expand to accommodate each line in an indirect file, and if there are too many lines before the system reaches a CTRL/C, the processor's memory area may become filled. Placing a CTRL/C every ten or so lines avoids this problem.

Some commands normally require a response from you as they execute. The INITIALIZE command, for example, prints the *Are you sure?* message and waits for you to type Y and a carriage return before it executes. The DELETE command also requests confirmation from you before it deletes a file. There are three ways to control interaction with the executing command. One way is to use the /NOQUERY option on each command that allows it. This option suppresses the confirmation messages entirely when you use the command in an indirect file.

Another method of interacting applies to a command like DELETE. This command can have a variable number of confirmation queries, especially if you use a wildcard in the file specification. This type of command accepts your responses directly from the terminal and allows you to make a decision before deleting each file. However, in this case the indirect file cannot operate unattended.

There is yet another way to deal with commands that require a response from you. Both the INITIALIZE and LINK commands have options that cause the system to prompt you for data. This section describes two methods of responding to these prompts, where more than just a Y response is required. The INITIALIZE command with the /VOLUMEID option permits you to specify a volume ID and owner name for a device. You can place your responses in the indirect file, as this example shows:

```
INITIALIZE/NOQUERY/VOLUMEID DT:
TAPE6
PAYROLL
```

You can change the indirect file so that the prompts appear on the console terminal and you can type your responses there:

```
INITIALIZE/NOQUERY/VOLUMEID DT:
^C
```

The ^C informs the system that the responses are to be entered at the terminal. Execution of the indirect file pauses until you enter the responses.

Similarly, the LINK command lets you specify some data either in the indirect file or from the console terminal. The following example contains the response to the TRANSFER prompt.

```
LINK/TRANSFER MYPROG,ODT
0,ODT
```

You can specify the same information interactively, as this example shows:

```
LINK/TRANSFER MYPROG,ODT
^C
```

The ^C informs the system that the response to the prompt is to be entered at the terminal. Execution of the indirect file pauses until you enter your response.

You can specify overlays to the LINK command by either of these two methods. The following indirect file links an overlaid program consisting of a root module and four overlay modules that reside in two overlay segments.

```
LINK/PROMPT ROOT
OVR1/O:1
OVR2/O:1
OVR3/O:2
OVR4/O:2//
```

Note in the above example that two slashes (//) terminate the module list. You can also enter all or part of the overlay information interactively, as this example shows:

```
LINK/PROMPT ROOT
OVR1/O:1
^C
```

The ^C informs the system that more overlay information is to be entered from the terminal. Execution of the indirect file pauses when the system requires the information. Respond to the asterisk prompt by entering the overlay information. Terminate the last overlay line with two slashes (//). Execution of the indirect file then proceeds. Chapter 11 describes the LINK program and explains how to use overlays.

If you need to link more than six modules, you can specify the extra modules on the next line in the indirect file, as this example shows:

```
LINK/PROMPT FIL1,FIL2,FIL3,FIL4,FIL5,FIL6
FIL7,FIL8//
```

Or, you can enter the extra modules from the terminal:

```
LINK/PROMPT FIL1,FIL2,FIL3,FIL4,FIL5,FIL6
^C
```

Execution of the indirect file pauses until you enter the remaining module names. Remember to follow the last name with two slashes (//).

You can include comments in an indirect file to help you document your work. These comments do not print on the console terminal when the indirect file executes. You begin each line of comment with an exclamation point (!). The system ignores any characters it finds between the exclamation point and the end of the current line. The following example shows an indirect file that contains comments.

```
!INDIRECT FILE
DATE                    !PRINT DATE
TIME                    !PRINT TIME
RENAME *.MAC *.BAK      !SAVE .MAC FILES
@PROCES                 !CALL ANOTHER INDIRECT FILE
DIRECTORY               !LIST DIRECTORY OF DK:
```

NOTE

You cannot place in indirect files responses to prompts that result in destruction of data. For example, you cannot use the INITIALIZE command followed by a Y on the following line in an indirect file. Commands like INITIALIZE and DELETE require responses that you must enter at the terminal. (You can avoid the need for a response by using the /NOQUERY option.)

4.3.2 Executing Indirect Files

You can execute indirect files under the SJ monitor, or in the background area under the FB or XM monitor.

To execute an indirect file, specify a command string according to the following syntax:

@filespec

where:

@	is the monitor command that indicates an indirect file
filespec	represents the name and file type of the indirect file, as well as the device on which it is stored. The default file type is .COM

If you omit the device specification, DK: is assumed. If you specify any other block-replaceable device, the monitor automatically loads the handler for that device. It is conventional to type the indirect file command directly in response to the monitor's prompt, as this example shows:

```
.@INDCT
```

However, you can place the indirect command anywhere in a keyboard monitor command string, as long as it is the last element in the string, not including comments. For example:

```
.DELETE/NOQUERY @INDCT!COMMENTS
```

This is a valid command string. The first line of the file should contain the list of files to be deleted. In the example above, assume the first line of the indirect file is:

```
*.BAK
```

This is the command that will actually execute:

```
DELETE/NOQUERY *.BAK
```

Check your indirect file carefully for errors before you execute it. When the monitor or any program that has control of the system encounters an illegal command line, or if an execution error of any kind occurs, that particular line does not execute properly. Execution of the indirect file does proceed, however, until any program that may be running relinquishes control to the monitor. Be careful of this if you run a system utility program in an indirect file, as this example shows:

```
R PIP
DX1:*.*=DX0:*.*
DX0:*.MAC/D
^C
PRINT DX0:*.LST
```

If device DX1: becomes full before all the files from DX0: are copied to it, the second line of the indirect file does not execute completely. Execution then passes to the next line and the system deletes all MACRO files from DX0:. The ^C returns control to the monitor, which aborts the rest of the indirect file. This example shows that it is possible to destroy files accidentally because of the way indirect files execute. To be safe, use only keyboard mon-

itor commands in an indirect file. This way the monitor regains control after each operation and can abort the indirect file as soon as it detects an error. A better way to perform the same operations as the indirect file shown above is as follows:

```
COPY DX0:*.* DX1:*.*
DELETE DX0:*.MAC
PRINT DX0:*.LST
```

You can use the SET ERROR command, described in Section 4.4, to define the severity of error that causes an indirect file to stop executing.

Normally, as each line of an indirect file executes, it echoes on the console terminal so that you can observe the progress of the job. However, you can use the SET TT QUIET command, described in Section 4.4, to suppress this printout. In this case, only the prompting messages, if any, print. You can stop execution of an indirect file at any time by typing two CTRL/C characters. Control returns to the monitor and you can enter a new command. You can also abort the indirect file by typing a single CTRL/C in response to a query or prompt. If you use an indirect file to execute a MACRO program, read the appropriate section in the *RT–11 Programmer's Reference Manual* to learn about certain restrictions on using the .EXIT call with indirect files.

You can call another indirect file from within an indirect file. This procedure is called nesting. Restrict nesting to three levels of indirect files (see the *RT–11 Installation and System Generation Guide* for details on selecting the indirect file nesting depth). The following example shows two-level nesting. Assume a programmer types this command at the console terminal in response to the monitor's prompt:

```
@FIRST
```

The file FIRST.COM contains these lines:

```
DATE
TIME
COPY *.MAC *.BAK
@SECOND
PRINT C
DIRECTORY/PRINTER DK:
DELETE/NOQUERY *.MAC
```

When this file executes it calls another indirect file, SECOND.COM, which contains this line:

```
MACRO/CROSSREFERENCE A+B+C/LIST
```

When file SECOND.COM finishes executing, control returns to file FIRST.COM, at the line following the indirect file specification. FIRST.COM then prints the contents of the file C.LST on the line printer, followed by a directory listing of device DK:. Then control returns to the monitor at the console terminal.

4.3.3 Startup Indirect Files

Section 3.1 introduced the startup indirect command files: STARTS.COM (for SJ), STARTF.COM (for FB), and STARTX.COM (for XM). Each monitor automatically invokes its own indirect command file when you bootstrap the system, and you can modify these files to perform standard system configurations. Since many of the system parameters are reset by a bootstrap operation (see the SET command, Section 4.4), you should use the startup indirect files to set the system parameters you normally use. For example, if you use the FB monitor and have a visual display console terminal that supports hardware tabs, add the SET TT: SCOPE and SET TT: TAB commands to the file STARTF.COM. You could also include a SET TT: QUIET command at the beginning of STARTF.COM and a SET TT: NOQUIET command at the end to suppress extra type-out at bootstrap time. If you have a list of commands that you need to execute, regardless of the monitor you bootstrap, include these commands in a separate indirect file, such as COMMON.COM, and invoke this file from all three startup indirect files. The following example shows a typical STARTF.COM file.

```
SET  TT:  QUIET                    !TURN OFF TTY PRINTING
SET  TT:  SCOPE
SET  TT:  TAB
@COMMON                            !PERFORM COMMON OPERATIONS
SET  TT:  NOQUIET                  !TURN ON TTY PRINTING
```

If you use BATCH frequently, use a startup indirect file to assign devices and load handlers. You can also use the startup indirect files to run your own programs, set the date, or do other housekeeping chores.

4.4 Keyboard Monitor Commands

The keyboard monitor commands are your means of communicating with the system and controlling the monitor. This section lists the keyboard monitor commands in alphabetical order. Each command description includes the command syntax, a table of valid options, and some sample command lines, as well as a general discussion of how to use the command.

You can type almost all the commands to any of the three monitors. The exceptions are FRUN, SRUN, SUSPEND, and RESUME. These are not valid for the SJ monitor because they apply to foreground programs.

Any reference to the background program applies also to the program running under the SJ monitor. Any reference to FB operation also applies to the XM operation.

NOTE

Unless noted otherwise, all numeric values you supply to keyboard commands should be in decimal.

If you make a mistake in a command line, or if the system cannot perform the action you request, an error message prints on your terminal. The error message indicates which error occurred; see the *RT–11 System Message Manual* for a more complete description of the error and for the recommended action to take. The error message also indicates which system utility program detected the error. For example, if your keyboard monitor command line contains a syntax error, the keyboard monitor prints an error message. If the utility program the keyboard monitor invokes cannot execute a command, that utility prints the error message.

RT–11 permits you to remove some of the monitor commands at system generation time. If you type a command that is not part of your system, the system prints an error message.

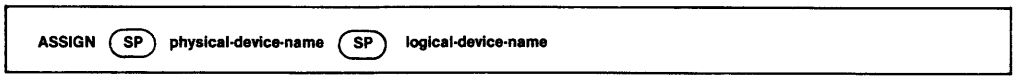
The APL command invokes the APL interpreter.

```
APL
```

Because APL has its own command language, the APL command accepts no options and no file specifications. For information on using the APL interpreter, see the *APL–11 Programmer's Reference Manual.*

ASSIGN

The ASSIGN command associates the logical name you specify with a physical device.



In the command syntax illustrated above, *physical-device-name* represents the RT–11 standard permanent name that refers to a particular device that is installed on your system. Table 3–1 contains a list of these names. The term *logical-device-name* represents an alphanumeric name, from one to three characters long, that you assign to a particular device. Note that you can not use spaces or tabs in the logical device name. If you type ASSIGN, followed by a carriage return, the system prompts: *Physical device name?*. If you follow the physical device name with a carriage return, the system prompts: *Logical device name?*.

If the logical device name you supply is already associated with a physical device, the system disassociates the logical name from that physical device and assigns it to the current device. You can assign only one logical name with each ASSIGN command, but you can use several ASSIGN commands to assign different logical names to the same device. You can also use the ASSIGN command to assign FORTRAN logical units to physical devices (see the *RT–11/RSTS/E FORTRAN IV User's Guide*).

The ASSIGN command simplifies programming. When you write a program, for example, you can request input from a device called INP: and direct output to a device called OUT:. When you are ready to execute the program, you can assign those logical names to the physical devices you need to use for that job. The ASSIGN command is especially helpful when a program refers to a device that is not available on a certain system; the ASSIGN command allows you to direct input and ouput to an available device.

Note that BA and SY are always invalid as logical device names.

The following command, for example, causes data that you write to device LST: to print on the line printer.

```
.ASSIGN LP: LST:
```

If your program attempts to access a device by using a logical name (such as LST:) and you do not issue an appropriate ASSIGN command, an error occurs in the program.

The following command redirects printer output to the terminal.

```
.ASSIGN TT: LP:
```

The command shown above illustrates how you can run a program that specifically references LP: without using a line printer.

The next command redefines the default file device.

```
. ASSIGN RK1: DK:
```

If you supply a file specification and omit the device name, it now defaults to RK1:. Note that this does not affect the default system device, SY:.

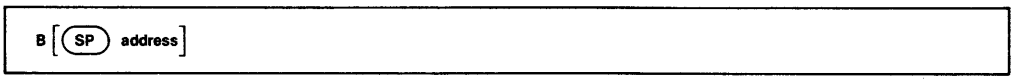
The last example is typical for a system that uses a dual-drive diskette device. Several users can share the same system software on DX0: and maintain their own data files on diskettes that they run in drive 1. When you use the following command, references to files without an explicit device name automatically access DX1:.

```
. ASSIGN DX1: DK:
```

Use the SHOW command to display logical device name assignments on the terminal.

B

The B (Base) command sets a relocation base. To obtain the address of the location to be referenced in a subsequent Examine or Deposit command, the system adds this relocation base to the address you specify.



In the command syntax shown above, *address* represents an octal address that the system uses as a base address for subsequent Examine and Deposit commands. If the address you supply is an odd number, the system decreases it by one to make the address even. Note that if you do not specify an address, this command sets the base to zero.

Use the B command when using the Examine and Deposit commands to reference linked modules that you have loaded into memory with the GET command. (Note that the Base command has no effect on program execution.) The system adds the current base address to the value you supply in an Examine or Deposit command. You can set the current base address to the address where a particular module is loaded. Then you can use the relocatable addresses printed in the assembler, compiler, or map listing of that module to reference locations within the module.

The following command sets the base to 0.

```
.B
```

The next two commands both set the base to 1000.

```
.B 1000
.B 1001
```

BASIC

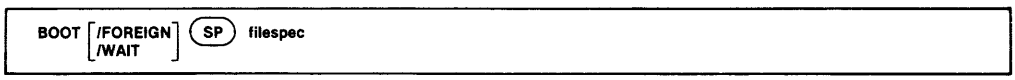
The BASIC command invokes the BASIC language interpreter.

```
BASIC
```

Because BASIC has its own command language, the BASIC command accepts no options and no file specifications. For information on using the BASIC interpreter, see the *BASIC–11 Language Reference Manual.*

BOOT

The BOOT command directs a new monitor to take control of the system. It can also read into memory a new copy of the monitor that is currently controlling the system.



In the command syntax illustrated above, *filespec* represents the device or monitor file to be bootstrapped. If you omit *filespec,* the system prompts you with *Device or file?*. The BOOT command can perform either of two operations: (1) a hardware bootstrap of a specific device, or (2) a direct bootstrap of a particular monitor file that does not use the bootstrap blocks on the device. When you bootstrap a volume, make sure that the appropriate device handler is present on that volume.

To perform a hardware bootstrap, specify only a device name in the command line. The following devices are valid for this operation:

DT0:-DT7:	DX0:-DX1:
RK0:-RK7:	PD0:-PD1:
RF:	DD0:-DD1:
SY:	DL0:-DL3:
DK:	DY0:-DY1:
DP0:-DP7:	DM0:-DM7:
DS0:-DS7:	

You can also boot any of the above storage volumes by specifying its logical name, if assigned (see the ASSIGN command). The hardware bootstrap operation gives control of the system to the monitor whose bootstrap is written on the device. (You can change this monitor by using the COPY/BOOT command.) This example bootstraps the single-job monitor, RT11SJ, whose bootstrap information is written on device DK:.

```
.BOOT DK:

RT-11SJ  V04.00
```

To bootstrap a particular monitor file, specify that file name and the device on which it is stored, if necessary, in the command line. SY: is the default device, and .SYS is the default file type.

You can use the BOOT command to alternate between the single-job and foreground/background monitors. When you use the BOOT command to change monitors you do not have to reenter the date and time. The system clock, however, may lose a few seconds during a reboot. The next example bootstraps the foreground/background monitor on device SY:, which is currently RK0:.

```
.BOOT RT11FB

RT-11FB  V04.00
```

/FOREIGN Use this option to boot a pre-version 4 volume or a non-RT–11 system. You may not specify a file name with /FOREIGN. The /FOREIGN option does not preserve the date or time.

/WAIT The /WAIT option is useful if you have a single-disk system. When you use this option, the system initiates the BOOT procedure but then pauses and waits for you to mount the volume you want to bootstrap. When the system pauses, it prints *Mount input volume in <device>; Continue?* at the terminal, where *<device>* represents the device into which you mount the volume. Mount the volume you want to bootstrap, then type Y followed by a carriage return.

The following sample command line boots an RK05 disk:

```
.BOOT/WAIT RK0:
Mount input volume in RK0:; Continue? Y
```

CLOSE

The CLOSE command closes and makes permanent all output files that are currently open in the background job.

```
CLOSE
```

The CLOSE command accepts no options or arguments.

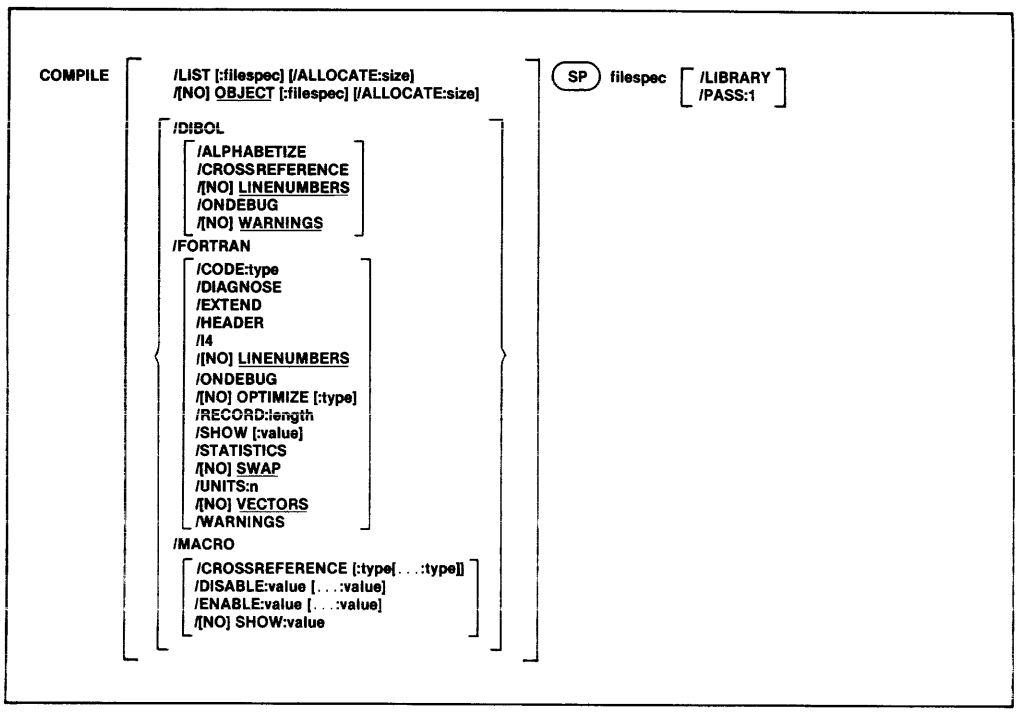
You can use the CLOSE command to make tentative open files permanent; otherwise, they do not appear in a normal directory listing and the space associated with the files is available for reuse. The CLOSE command is particularly useful after you type a CTRL/C to abort a background job. You can also use it after an unexpected program termination to preserve any new files that were being used by the terminated program. Note that the CLOSE command has no effect on a foreground job and that you cannot use CLOSE on files opened on magnetic tape or cassette.

The CLOSE command does not work if your program defines new input or output channels (with the .CDFN programmed request). Because CTRL/C or .EXIT resets channel definitions, the CLOSE command has no effect on channels it does not recognize.

The following example shows how the CLOSE command makes temporary files permanent.

```
•R PROG
    •
    •
    •
^C ^C
•CLOSE
```

The COMPILE command invokes the appropriate language processor to assemble or compile the files you specify.



In the command line shown above, *filespecs* represents one or more files to be included in the assembly or compilation. The default file types for the output files are .LST for listing files and .OBJ for object files. The defaults for input files depend on the particular language processor involved and include .MAC for MACRO files, .FOR for FORTRAN files, and .DBL for DIBOL files.

To compile (or assemble) multiple source files into a single object file, separate the files by plus (+) signs in the command line. Unless you specify otherwise, the system creates an object file with the same name as the first input file and gives it an .OBJ file type. To compile multiple files in independent compilations, separate the files by commas (,) in the command line. This generates a corresponding object file for each set of input files. You can combine up to six files for a compilation producing a single object file.

Language options are position-dependent — that is, they have different meanings depending on where you place them in the command line. Options that qualify a command name apply across the entire command string. Options that follow a file specification apply only to the file (or group of files separated by plus signs) that they follow in the command string.

You can specify the entire COMPILE command as one line, or you can rely on the system to prompt you for information. The COMPILE command prompt is *Files?*.

There are three ways to establish which language processor the COMPILE command invokes.

1. Specify a language-name option, such as /MACRO, which invokes the MACRO assembler.
2. Omit the language-name option and explicitly specify the file type for the source files. The COMPILE command then invokes the language processor that corresponds to that file type. Specifying the file SOURCE.MAC, for example, invokes the MACRO assembler.
3. Let the system choose a file type of .MAC, .DBL, or .FOR for the source file you name. To do this, the handler for the device you specify must be loaded. If you specify DX1:A and the DX handler is loaded, the system searches for source files A.MAC and A.DBL, in that order. If it finds one of these files, the system invokes the corresponding language processor. If it cannot find one of these files, or if the device handler associated with the input file is not resident, the system assumes a file type of .FOR and invokes the FORTRAN compiler.

If the language processor selected as a result of one of the procedures described above is not on the system device (SY:), the system issues an error message.

The following sections explain the options you can use with the COMPILE command.

/ALLOCATE:size Use this option with /LIST or /OBJECT to reserve space on the device for the output file. The argument *size* represents the number of blocks of space to allocate. The meaningful range for this value is from 1 to 32767. A value of –1 is a special case that creates the largest file possible on the device.

/ALPHABETIZE Use this option with /DIBOL to alphabetize the entries in the symbol table listing. This is useful for program maintenance and debugging.

/CODE:type Use this option with /FORTRAN to produce object code that is designed for a particular hardware configuration. The argument *type* represents a three-letter abbreviation for the type of code to produce. The legal values are: EAE, EIS, FIS, and THR. See the *RT–11/RSTS/E FORTRAN IV User's Guide* for a complete description of the types of code and their function.

/CROSSREFERENCE[:type[...:type]] Use this option with /MACRO or /DIBOL to generate a symbol cross-reference section in the listing. This information is useful for program maintenance and debugging. Note that the system does not generate a listing by default. You must also specify /LIST in the command line to get a cross-reference listing.

With /MACRO, this option takes an optional argument. The argument *type* represents a one-character code that indicates which sections of the cross-reference listing the assembler should include. See the MACRO command in this chapter for a summary of valid arguments and their meaning.

/DIAGNOSE Use this option with /FORTRAN to help analyze an internal compiler error. /DIAGNOSE expands the crash dump information to include internal compiler tables and buffers. Submit the diagnostic printout to DIGITAL with an SPR form. The information in the listing can help the DIGITAL programmers locate the compiler error and correct it.

/DIBOL This option invokes the DIBOL language processor to compile the associated files.

/DISABLE:value[...:value] Use this option with /MACRO to specify a .DSABL directive. See the MACRO command in this chapter for a summary of the arguments and their meaning. See the *PDP–11 MACRO Language Reference Manual* for a description of the directive and a list of all legal values.

/ENABLE:value[...:value] Use this option with /MACRO to specify an .ENABL directive. See the MACRO command in this chapter for a summary of the arguments and their meaning. See the *PDP–11 MACRO Language Reference Manual* for a description of the directive and a list of all legal values.

/EXTEND Use this option with /FORTRAN to change the right margin for source input lines from column 72 to column 80.

/FORTRAN This option invokes the FORTRAN language processor to compile the associated files.

/HEADER Use this option with /FORTRAN to include in the printout a list of options that are currently in effect.

/I4 Use this option with /FORTRAN to allocate two words for the default integer data type (FORTRAN uses only one-word integers) so that it takes the same physical space as real variables.

/LIBRARY Use this option with /MACRO to identify a macro library file; use it only after a library file specification in the command line. The MACRO assembler looks first to any MACRO libraries you specify before going to the default system macro library, SYSMAC.SML, to satisfy references (made with the .MCALL directive) from MACRO programs. In the example below, the two files A.FOR and B.FOR are compiled together, producing B.OBJ and B.LST. The MACRO assembler assembles C.MAC, satisfying .MCALL references from MYLIB.MAC and SYSMAC.SML. It produces C.OBJ and C.LST.

```
.COMPILE A+B/LIST/OBJECT,MYLIB/LIBRARY+C.MAC/LIST/OBJECT
```

/LINENUMBERS Use this option with /DIBOL or /FORTRAN to include internal sequence numbers in the executable program. These numbers are especially useful in debugging programs. This is the default operation.

/NOLINENUMBERS Use this option with /DIBOL or /FORTRAN to suppress the generation of internal sequence numbers in the executable program. This produces a smaller program and optimizes execution speed. Use this option to compile only those programs that are already debugged; other-

wise the line numbers in DIBOL or FORTRAN error messages are difficult to interpret.

/LIST[:filespec] You must specify this option to produce a compilation or assembly listing. The /LIST option has different meanings depending on its position in the command line.

If you specify /LIST without a file specification in the list of options that immediately follows the command name, the system generates a listing that prints on the line printer. If you follow /LIST with a device name, the system creates a listing file on that device. If the device is a file-structured device, the system stores the listing file on that device, assigning it the first input file name and a .LST file type. The following command produces a listing on the terminal:

```
.COMPILE/LIST:TT:  A.FOR
```

The next command creates a listing file called A.LST on RK3:.

```
.COMPILE/LIST:RK3: A.MAC
```

If the /LIST option contains a name and file type to override the default of .LST, the system generates a listing file with that name. The following command, for example, compiles A.FOR and B.FOR together, producing files A.OBJ and FILE1.OUT on device DK:.

```
.COMPILE/FORTRAN/LIST:FILE1.OUT A+B
```

Another way to specify /LIST is to type it after the file specification to which it applies. To produce a listing file with the same name as a particular input file, you can use a command similar to this one:

```
.COMPILE/DIBOL A+B/LIST:RK3:
```

The command shown above compiles A.DBL and B.DBL together, producing files DK:A.OBJ and RK3:B.LST. If you specify a file name on a /LIST option following a file specification in the command line, it has the same meaning as when it follows the command. The following two commands have the same results:

```
.COMPILE/MACRO A/LIST:B

.COMPILE/MACRO/LIST:B A
```

Both the commands shown above generate as output files A.OBJ and B.LST on device DK:.

Remember that file options apply only to the file (or group of files that are separated by plus signs) they follow in the command string. For example:

```
.COMPILE A.MAC/LIST,B.FOR
```

This command compiles A.MAC, producing A.OBJ and A.LST on DK:. It also compiles B.FOR, producing B.OBJ on DK:. However, it does not produce any listing file for the compilation of B.FOR.

/MACRO This option invokes the MACRO assembler to assemble the associated files.

/OBJECT[:filespec] Use this option to specify a file name or device for the object file. Because the COMPILE command creates object files by default, the following two commands have the same meaning:

```
COMPILE/FORTRAN A

COMPILE/FORTRAN/OBJECT A
```

Both commands compile A.FOR and produce A.OBJ as output. The /OBJECT option functions like the /LIST option; it can be either a command or a file qualifier.

As a command option, /OBJECT applies across the entire command string. The following command, for example, assembles A.MAC and B.MAC separately, creating object files A.OBJ and B.OBJ on RK1:.

```
COMPILE/OBJECT:RK1: (A,B).MAC
```

Use /OBJECT as a file option to create an object file with a specific name or destination. The following command compiles A.DBL and B.DBL together, creating files B.LST and B.OBJ.

```
COMPILE/DIBOL A+B/LIST/OBJECT
```

/NOOBJECT Use this option to suppress creation of an object file. As a command option, /NOOBJECT suppresses all object files; as a file option, it suppresses only the object file produced by the related input files. In this command, for example, the system compiles A.FOR and B.FOR together, producing files A.OBJ and B.LST. It also compiles C.DBL and produces C.LST, but it does not produce C.OBJ.

```
COMPILE A.FOR+B.FOR/LIST,C.DBL/NOOBJECT/LIST
```

/ONDEBUG Use this option with /DIBOL to include a symbol table in the object file. You can then use a debugging program to find and correct errors in the object file.

Use /ONDEBUG with FORTRAN to include debug lines (those that have a D in column 1) in the compilation. You do not, therefore, have to edit the file to include these lines in the compilation or to logically remove them. This option means that you can include messages, flags, and conditional branches to help you trace program execution and find errors.

/OPTIMIZE[:type] Use this option with /FORTRAN to enable certain options that optimize object code for various conditions. The argument *type* represents the three-letter code for the type of optimization to enable. Table 4–4 summarizes the codes and their meaning. This option is not available in version 2.5 of the FORTRAN compiler.

/NOOPTIMIZE[:type] Use this option with /FORTRAN to disable certain options that optimize object code for various conditions. The argument *type* represents the three-letter code for the type of optimization to disable. Table

4–4 summarizes the codes and their meaning. This option is not available in version 2.5 of the FORTRAN compiler.

/PASS:1 Use this option with /MACRO on a prefix macro file to process that file during pass 1 of the assembly only. Using this option means that you can assemble a source program together with a prefix file that contains only macro definitions, because these definitions do not need to be redefined in pass 2 of the assembly. The following command assembles a prefix file and a source file together, producing files PROG1.OBJ and PROG1.LST.

```
.COMPILE/MACRO PREFIX/PASS:1+PROG1/LIST/OBJECT
```

/RECORD:length Use this option with /FORTRAN to override the default record length of 132 characters for ASCII sequential formatted input and output. The meaningful range for the argument *length* is from 4 to 4095.

/SHOW:value Use this option with /FORTRAN to control FORTRAN listing format. The argument *value* represents a code that indicates which listings the compiler is to produce. Table 4–5 summarizes the codes and their meaning.

Use this option with /MACRO to specify any MACRO .LIST directive. Table 4–12 summarizes the valid arguments and their meaning. The *PDP–11 MACRO Language Reference Manual* explains how to use these directives.

/NOSHOW:value Use this option with MACRO to specify any MACRO .NLIST directive. Table 4–12 summarizes the valid arguments and their meaning. The *PDP–11 MACRO Language Reference Manual* explains how to use these directives.

/STATISTICS Use this option with /FORTRAN to include compilation statistics in the listing, such as amount of memory used, amount of time elapsed, and length of the symbol table.

/SWAP Use this option with /FORTRAN to permit the USR (User Service Routine) to swap over the FORTRAN program in memory. This is the default operation.

/NOSWAP Use this option with /FORTRAN to keep the USR resident during execution of a FORTRAN program. This may be necessary if the FORTRAN program uses some of the RT–11 system subroutine calls (see the *RT–11 Programmer's Reference Manual*). If the program frequently updates or creates a large number of different files, making the USR resident can improve program execution. However, the cost for making the USR resident is 2K words of memory.

/UNITS:n Use this option with FORTRAN to override the default number of logical units (6) to be open at one time. The maximum value you can specify for *n* is 16.

/VECTORS This option directs FORTRAN to use tables to access multidimensional arrays. This is the default mode of operation.

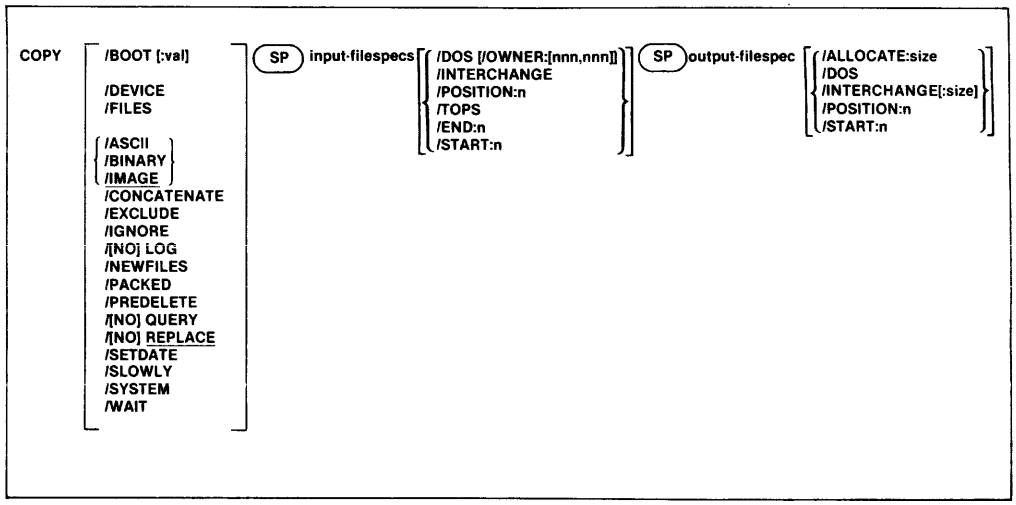
/NOVECTORS This option directs FORTRAN to use multiplication operations to access multidimensional arrays.

/WARNINGS Use this option to include warning messages in DIBOL or FORTRAN compiler diagnostic error messages. These messages call certain conditions to your attention but do not interfere with the compilation. This is the default operation for DIBOL.

/NOWARNINGS Use this option with /DIBOL or /FORTRAN to suppress warning messages during compilation. These messages are for your information only; they do not affect the compilation. This is the default operation for FORTRAN.

COPY

The COPY command performs a variety of file transfer and maintenance operations.



The COPY command transfers:

- one file to another file
- a number of files to a single file by concatenation
- the contents of a device to another device
- the contents of a bootstrap to a device
- the contents of a device to a file and vice versa

In the command syntax shown above, *input-filespecs* represents the data to copy. The *input-filespec* can be a device name, if you use the /DEVICE option. Otherwise, you can specify as many as six files for input. *Output-filespec* represents the device or file to receive the data. You can specify only one output device or file.

Normally, commas separate the input files if you specify more than one. However, you can separate them by plus (+) signs if you want to combine them, as the following example shows:

```
.COPY A.FOR+B.FOR C.FOR
```

This command combines DK:A.FOR with DK:B.FOR and stores the results in DK:C.FOR.

Note that because of the file protection feature, you cannot execute any COPY operations that result in the deletion of a protected file. For example, you cannot copy a file from one volume to another if a protected file of the same name already exists on the output volume.

You can use wildcards in the input or output file specification of the command. However, the output file specification cannot contain embedded wildcards. Note that for all operations except CONCATENATE, if you use a wildcard in the input file specification, the corresponding output file name or file type must be an asterisk (*). This example uses wildcards correctly:

```
.COPY A%B.MAC *.BAK
```

In the CONCATENATE operation, the output specification must represent a single file. Therefore, no wildcards are allowed.

You can enter the COPY command as one line, or you can rely on the system to prompt you for information. If you type COPY followed by a carriage return, the system prompts *From?*. If you type the input specification followed by a carriage return, the system prompts *To?*.

The system has a special way of handling system (.SYS) files and files that cover bad blocks (.BAD files). The system requires you to use the /SYSTEM option when you need to copy system files. You cannot copy system files simply by placing wildcards in file specifications. To copy a .BAD file, you must specify it by explicitly giving its file name and file type. Since .BAD files cover bad blocks on a device, you usually do not need to copy, delete, or otherwise manipulate these files. You can copy protected files (see RENAME), but you cannot copy the protection status of a protected file (except with the COPY/DEVICE command).

NOTE

If you transfer files to a storage volume that has never been initialized with RT–11, a system failure may result.

The following sections describe the COPY command options and include command examples.

/ALLOCATE:size Use this option after the output file specification to reserve space on the device for the output file. The argument *size* represents the number of blocks of space to allocate. The meaningful range for this value is from 1 to 32767. A value of –1 is a special case that creates the largest file possible on the device.

/ASCII This option copies files in ASCII mode, ignoring and eliminating nulls and rubout characters. It converts data to the ASCII 7-bit format and treats CTRL/Z (32 octal) as the logical end-of-file on input. Files that consist of ASCII-format data include source files you create with the editor, map files, and list files. The following example copies a FORTRAN source program from DX0: to DX1:, giving it a new name, and reserving 50 blocks of space for it.

```
.COPY/ASCII DX0:MATRIX.FOR DX1:TEST.FOR/ALLOCATE:50
```

/BINARY Use this option to copy formatted binary files, such as .OBJ files produced by the assembler or the FORTRAN compiler, and .LDA files produced by the linker. The system verifies checksums and prints a warning if a

checksum error occurs. If this happens, the copy operation does not complete. The following command copies a binary file from DK: to a diskette.

```
.COPY/BINARY ANALYZ.OBJ DX1:*.*
```

Note that you cannot copy library files with the /BINARY option because a checksum error occurs. Copy them in image mode, or when you are creating a bootable RX01 system while the current system is on an RX02.

/BOOT[:val] This option copies bootstrap information from a monitor and handler files to blocks 0 and 2 through 5 of a random-access volume, permitting you to use that volume as a system volume. The optional argument *val* represents a two-letter target system device name that you use when you are creating a bootable PDT system volume on a PDP–11, and vice versa. You can also use this notation to create a bootable RX01 system while the current system is on an RX02 diskette. Note that you cannot combine /BOOT with any other option, and that your input and output volume must be the same. Also note that you can name your monitor file any name you wish. When you perform this operation, you must have the correct device handler to go with the volume. For example, to create a bootable RK05 disk, you must have the handler file RK.SYS on that RK05.

To create a bootable system volume, follow the procedure below:

1. Initialize the volume, using the keyboard monitor command INITIALIZE. (Note that if the volume is an RK06/07 or an RL01/02, you should also use the /REPLACE option.)
2. Copy files onto the volume, using the COPY/SYSTEM command.
3. Write the monitor bootstrap onto the volume, using COPY/BOOT.

The following example creates a system diskette.

```
.INITIALIZE DX1:
DX1:/Initialize; Are you sure? Y

.COPY/SYSTEM DX0:*.* DX1:*.*
 Files copied:
DX0:RT11SJ.SYS to DX1:RT11SJ.SYS
DX0:DT.SYS     to DX1:DT.SYS
DX0:DX.SYS     to DX1:DX.SYS
DX0:TT.SYS     to DX1:TT.SYS
DX0:LP.SYS     to DX1:LP.SYS
DX0:DIR.SAV    to DX1:DIR.SAV
DX0:DUP.SAV    to DX1:DUP.SAV
DX0:ABC.MAC    to DX1:ABC.MAC
DX0:AAF.MAC    to DX1:AAF.MAC
DX0:CT.SYS     to DX1:CT.SYS
DX0:PIP.SAV    to DX1:PIP.SAV
DX0:MT.SYS     to DX1:MT.SYS
DX0:MM.SYS     to DX1:MM.SYS
DX0:COMB.DAT   to DX1:COMB.DAT
DX0:RT11FB.SYS to DX1:RT11FB.SYS

.COPY/BOOT DX1:RT11FB.SYS DX1:
```

The device names you can use for the optional argument *val* are PD, DD, DX, and DY. The following example creates a bootable system diskette for a PDT while the current system is on a PDP–11:

```
.COPY/BOOT:PD  DX0:RT11SJ.SYS  DX0:
```

The following example creates a bootable system diskette for a PDP–11 while the current system is on a PDT–11/150:

```
COPY/BOOT:DX  PD1:RT11SJ.SYS  PD1:
```

Note that the monitor file cannot reside on a block that contains a bad sector error (BSE) if you are doing bad block replacement. If this condition occurs, a boot error results when you bootstrap the system. In this case, move the monitor so that it does not reside on a block with a BSE error.

/CONCATENATE Use this option to combine several input files into a single output file. This option is particularly useful to combine several object modules into a single file for use by the linker or librarian. The following command combines all the .FOR files on DX1: into a file called MERGE.FOR on DX0:.

```
.COPY/CONCATENATE DX1:*.FOR DX0:MERGE.FOR
 Files copied:
DX1:A.FOR       to DX0:MERGE.FOR
DX1:B.FOR       to DX0:MERGE.FOR
DX1:C.FOR       to DX0:MERGE.FOR
```

Wildcards are illegal in the output file specification.

/DEVICE This option copies block for block the image of one device to another. This option copies all data from one disk to another without changing the file structure or the location of the files on the device. This is convenient in that the bootstrap blocks also remain unchanged. You can also copy disks that are not in RT–11 format, as long as they have no bad blocks. If the system encounters a bad block during the COPY/DEVICE operation, it prints an error message. However, it then retries the operation and performs the copy one block at a time. If only one error message prints, you can assume that the transfer completed correctly.

If one device is smaller than the other, the system copies only as many blocks as the smaller device contains. For example, if you copy a large volume to a smaller one, you copy the entire directory of the input volume, but not every file in the input volume. It is possible to copy blocks between disk and magtape, even though magtape is not a random-access device. The data is stored on tape formatted in 1K-word blocks. Because magtape is not file-structured, there is room for only one disk image on a magtape. The following command copies an image of DX0: to DX1:.

```
.COPY/DEVICE DX0: DX1:
DX1:/Copy;  Are you sure?  Y
```

Respond to the query message by typing Y and a carriage return. Any response not beginning with Y cancels the command and the COPY operation does not proceed.

NOTE

The COPY command does not copy track 0 of diskettes. However, this restriction has no impact on any copy operations if your diskette was supplied by DIGITAL.

/DOS Use this option to transfer files between RSTS/E or DOS–11 format and RT–11 format. The option must appear in the command line after the file to which it applies. Valid input devices are DECtape and RK05; the only valid output device is DECtape. The only other options allowed with /DOS are /ASCII, /BINARY, /IMAGE, and /OWNER:[nnn,nnn]. The following command transfers a BASIC source file from a DOS–11 disk to an RT–11 disk.

```
.COPY RK:PROG.BAS/DOS/OWNER:[200,200] SY:*.*
```

The next command copies a memory image file from an RT–11 disk to a RSTS/E format DECtape.

```
.COPY DUMP.SAV DT:*.*/DOS
```

/END:n Use with /START:n and /DEVICE to specify the last block of the volume you are copying. The /END:n notation must follow the input file specification. The argument *n* represents a decimal block number. The following example copies blocks 0 to 500 from RK0: to RK1:, starting at block 501, in a file named ADAM.MAC:

```
.COPY RK0:/START:0/END:500  RK1:ADAM.MAC/START:501
```

/EXCLUDE This option copies all the files on a device except the ones you specify. The following command copies all files from DX0: to DX1: except .OBJ and .SAV files.

```
.COPY/EXCLUDE DX0:(*.OBJ,*.SAV) DX1:*.*
```

/FILES Use with /DEVICE to copy a volume to a file on another volume or vice versa. If you use a magtape or cassette for the input volume, you must specify a file name with the input volume. This operation is useful if you wish to make several copies of a volume that is on a slow device. You can copy the volume as a file onto a volume that is on a faster device, and then proceed to make copies. Note that when you copy a file to a volume, the bootstrap and directory of the output volume are replaced by the equivalent blocks of the input file.

The following example copies diskette DX0: to DL1: as file FLOPPY.BAK:

```
.COPY/DEVICE/FILES DX0:  DL1:FLOPPY.BAK
```

The following example copies file DECTAP.BAK to DD0:

```
.COPY/DEVICE/FILES  DECTAP.BAK  DD0:
```

/IGNORE Use this option to ignore errors during a copy operation. /IGNORE forces a single-block data transfer, which you can invoke at any other time with the /SLOWLY option. Use /IGNORE if an input error

occurred when you tried to perform a normal copy operation. This procedure can sometimes recover a file that is otherwise unreadable. If there is still an error, an error message prints on the terminal, but the copy operation continues. This option is invalid with /DOS, /TOPS, and /INTERCHANGE.

/IMAGE If you enter a command line without an option, or if you use the /IMAGE option, the copy operation proceeds in image mode. Use this method to transfer memory image files and any files other than ASCII or formatted binary. Note that you cannot transfer memory image files reliably to or from paper tape, or to the line printer or console terminal. You can image-copy ASCII and binary data with the following restrictions:

1. For ASCII data, there is no check for nulls.
2. For binary data, there is no checksum consideration.

This command copies a text file to a DECtape for storage:

```
.COPY LETTER.TXT DT0:*.*
```

The primary advantage to using /IMAGE is that it is faster than /ASCII and /BINARY.

/INTERCHANGE[:size] This option transfers data in interchange format between RT–11 block-replaceable devices and interchange diskettes that are compatible with IBM 3741 format. The option must appear in the command line after the file to which it applies. If the output file is to be in interchange format, you can specify the length of each record. The argument *size* represents the record length in characters. The following command transfers the RT–11 file WAIT.MAC from device DK: to device DX1: in interchange format, giving it the name WAIT.MA. The record length is set to 128 (decimal) bytes.

```
.COPY WAIT.MAC DX1:*.*/INTERCHANGE:128.
```

/LOG This option lists on the terminal the names of the files that were copied by the current command. Normally, the system prints a log only if there is a wildcard in the file specification. If you specify /QUERY, the system prints the name of each file and asks you for confirmation before the operation proceeds. In this case, the query messages replace the log, unless you specifically type /LOG/QUERY in the command line. The following example shows a copy command line and the resulting log.

```
.COPY/LOG DX1:FILE.MAC  DX0:FILE.MAC
 Files copied:
DX1:FILE.MAC   to DX0:FILE.MAC
```

/NOLOG This option prevents a list of the files copied from appearing on the terminal.

/NEWFILES Use this option in the command line if you want to copy only those files that have the current date. The following example shows a convenient way to back up all new files after a session at the computer.

```
.COPY/NEWFILES *.* DX1:*.*
 Files copied:
DK:A.FOR        to DX1:A.FOR
DK:B.FOR        to DX1:B.FOR
DK:C.FOR        to DX1:C.FOR
```

/OWNER:[nnn,nnn] Use this option with /DOS to represent a DOS–11 user identification code (UIC) for a DOS–11 input device. Note that the square brackets are part of the UIC; you must type them. The initial default for the UIC is [1,1].

/PACKED This option copies files in DECsystem–10, DOS, or interchange mode. You can use /PACKED on an input file specification with the /TOPS, /DOS, or /INTERCHANGE option to transfer files to RT–11 format.

/POSITION:n Use this option when you copy files to or from magtape or cassette. The /POSITION:n option lets you direct the tape operation; you can move the tape and perform an operation at the point you specify. For all operations, omitting the argument *n* has the same effect as setting *n* equal to 0 (*n* is interpreted as a decimal number). Since this option applies to the device and not to the files, you can specify one /POSITION:n option for the output file and one for the input files.

For magtape read (copy from tape) operations, the /POSITION:n option initiates these procedures:

1. If *n* is 0:
   The tape rewinds and the handler searches for the file you specify. If you specify more than one file, the tape rewinds before each search. If the file specification contains a wildcard, the tape rewinds only once and then the handler copies all the appropriate files.

2. If *n* is a positive integer:
   The handler looks for the file at file sequence number *n*. If the file it finds there is the one you specify, the handler copies it. Otherwise, the it prints an error message. If you use a wildcard in the file specification, the handler goes to file sequence number *n* and then begins to look for the appropriate files.

3. If *n* is –1:
   The handler starts its search at the current position. Note that if the current position is not the beginning of the tape, it is possible that the file you specify will not be found, even though it does exist on the tape.

For magtape write (copy to tape) operations, the /POSITION:n option has this effect:

1. If *n* is 0:
   The tape rewinds before the handler copies each file. A warning message prints on the terminal if the handler finds another file on the tape with the same name and file type, and the handler does not copy the file.

2. If *n* is a positive integer:
   The handler goes to file sequence number *n* or to the logical end of tape,

whichever comes first. Then it enters the file you specify. If you specify more than one file, or if you use a wildcard in the file specification, the tape does not rewind before the handler writes each file, and the handler does not check for duplicate file names. If the handler finds the sequence number *n*, it creates a new logical end of tape. If there are any files with a sequence number greater than *n*, they are lost.

3. If *n* is –1:
   The handler goes to the logical end-of-tape and enters the file you specify. It does not rewind, and it does not check for duplicate file names.

4. If *n* is –2:
   The tape rewinds between each copy operation. The handler enters the file you specify at logical end-of-tape or at the first occurrence of a duplicate file name (but if the handler enters the file over the duplicate file, you lose everything after that file).

The handler also has special procedures for handling cassettes. For cassette read (copy from tape) operations, the /POSITION:n option initiates these procedures:

1. If *n* is 0:
   The cassette rewinds and the handler searches for the file you specify. If you specify more than one file, or if you use a wildcard in the file specification, the cassette rewinds before each search.

2. If *n* is a positive integer:
   The handler starts from the cassette's present position and searches for the file you specify. If the handler does not find the file you specify before it reaches the *n*th file from its starting position, it reads the *n*th file. Note that if the starting position is not the beginning of the tape, it is possible that the handler will not find the file you specify, even though it does exist on the tape.

3. If *n* is a negative integer:
   The cassette rewinds, then the handler follows the procedure outlined in step 2 above.

For cassette write (copy to tape) operations, the /POSITION:n has this effect:

1. If *n* is 0:
   The cassette rewinds and the handler writes the file you specify at the logical end-of-tape. The handler automatically deletes any file it finds that has the same name and file type as the file you specify.

2. If *n* is a positive integer:
   The handler starts from the cassette's present position and searches *n* files ahead, deleting along the way any file it finds that has the same name and file type as the file you specify. If the handler does not reach the logical end-of-tape before it reaches the *n*th file from its starting position, it enters the file you specify over the *n*th file and deletes any files beyond it on the tape. If the handler reaches the logical end-of-tape

before it reaches the *n*th file, it writes the file you specify at the end-of-tape position.

3. If *n* is a negative integer:
   The cassette rewinds, then the handler follows the same procedure outlined in step 2 above.

Chapter 7, Section 7.2.1, contains more detailed information about operations involving magtape and cassette.

/PREDELETE This option deletes a file on the output device that has the same name as a file you copy to that device. The system deletes the file on the output device before the copy occurs. Normally, the system deletes a file of the same name after the copy operation successfully completes. This option is useful for operations involving devices that have limited space, such as diskette. Be careful when you use the /PREDELETE option; if for any reason the input file is unreadable, the output file will already have been deleted and you are left with no usable version of the file. Cassette devices are valid for input files but not for output.

/QUERY If you use this option, the system requests confirmation from you before it performs the operation. /QUERY is particularly useful on operations that involve wildcards, when you may not be sure which files the system selected for an operation. The /QUERY option is valid on the COPY command only if both input and output are in RT-11 format. Note that if you specify /QUERY in a copy command line that also contains a wildcard in the file specification, the confirmation messages that print on the terminal replace the log messages that would normally appear. You must respond to a query message by typing Y (or anything that begins with a Y) and a carriage return. The system interprets any other response to mean NO, and it does not copy the file. The following example copies three of the four FOR files stored on DK: to DX1:.

```
.COPY/QUERY DK:*.FOR DX1:*.*

DK:A.FOR        to DX1:A.FOR      ? Y
DK:B.FOR        to DX1:B.FOR      ? Y
DK:C.FOR        to DX1:C.FOR      ? N
DK:DEMOF1.FOR   to DX1:DEMOF1.FOR? Y
```

/NOQUERY This option suppresses the confirmation message that the system prints for some operations, such as COPY/DEVICE. It also suppresses logging of file names if the command line contains a wildcard. You must explicitly type /LOG to obtain a list of the files copied.

/REPLACE This is the default mode of operation for the COPY command. If a file exists on the output device with the same name as the file you specify for output, the system deletes the duplicate file after the copy operation successfully completes.

/NOREPLACE This option prevents execution of the copy operation if a file with the same name as the output file you specify already exists on the output device. /NOREPLACE is valid only if both the input and output are in RT–11 format.

/SETDATE This option causes the system to put the current date on all files it transfers, unless the current system date is zero. Normally, the system preserves the existing file creation date when it copies a file block for block. This option is invalid for operations involving magtape and cassette, because the system always uses the current date for tape files.

/SLOWLY This option transfers files one block at a time. On some devices, a single-block transfer increases the chances of an error-free transfer. Use this option if a previous copy operation failed because of a read or write error.

/START[:n] Use with the /DEVICE option to specify the starting block and, with /END:n, to specify the last block of the disk you are copying. The /START:n notation must follow the input or output file specification. The argument *n* with both /START and /END represents a decimal block number.

You can use /START:n with the output file specification to specify the starting block number for the write operation on the output volume.

The following example copies blocks 500 to 550 of RK0: to RK1: starting at block 100:

```
.COPY RK0:/START:500/END:550  RK1:/START:100
```

If you do not supply a value with /START, the system assumes the first block on the volume. If you do not specify a value with /END, the system assumes the last block on the volume. Note that the first block of a file or volume is block 0.

/SYSTEM Use this option if you need to copy system (.SYS) files. If you omit this option, the .SYS files are excluded from all operations and a message is printed on the terminal to remind you.

/TOPS This option transfers files on DECsystem–10 DECtape to RT–11 format. The option must follow the input file specification. Note that DECtape is the only valid input device. You cannot perform this copy operation while a foreground job is running. Use /PACKED with /TOPS to convert from TOPS–10 7-bit ASCII format to standard PDP–11 byte ASCII format. The following command copies in ASCII format all the files named MODULE from the DECsystem–10 DECtape DT0: to RT–11 device RK0:.

```
.COPY/ASCII DT0:MODULE.*/TOPS RK0:*.*
```

/WAIT Use this option on systems that have only a single-disk drive, or on systems that have dual drive and the system volume is neither the input nor output volume. When you use this option, the system initiates execution of a command but then pauses and prints the message *Continue?*. At this time, you can remove the system disk and mount the disk on which you want the operation to take place. When the new disk is loaded, type a Y followed by a carriage return to resume the operation. When the operation completes, the system prints the *Continue?* message again. Mount the system volume and type a Y followed by a carriage return. The system then prints the keyboard

monitor prompt. Make sure PIP and DUP are on your system volume when you use the /WAIT option. The /WAIT option is valid with /DEVICE.

Single-Volume Operation

If you want to transfer a file between two storage volumes, and you have only one drive for that type of storage volume, follow the procedure below.

1. Enter a command string according to this general syntax:

   COPY/WAIT input-filespec output-filespec

   where *output-filespec* represents the destination device and file specification, and *input-filespec* represents the source device and file specification.

2. The system responds by printing the following message at the terminal.

   ```
   Mount input volume in <device>:  Continue?
   ```

   <device> represents the device into which you are to mount your input volume. Type a Y followed by a carriage return after you have mounted your input volume.

3. The system continues the copy procedure and prints the following message on the terminal:

   ```
   Mount output volume in <device>:  Continue?
   ```

4. After you have removed your input volume from the device, mount your output volume, then type Y followed by a carriage return.

5. Depending on the size of the file, the system may repeat the transfer cycle (steps 2 and 3) several times before the transfer is complete. When the transfer is complete, the system prints the following prompt at the terminal:

   ```
   Mount system volume in <device>:  Continue?
   ```

   When you mount your system volume and type a Y followed by a carriage return in response to the last instruction, you terminate the copy operation.

Double-Volume Operation

If you have a small disk system, you can use the /WAIT option for transferring files between two non-system volumes. The procedure for transferring files this way follows.

1. With your system volume mounted, enter a command according to the following general syntax:

   COPY/WAIT input-filespec output-filespec

   where *output-filespec* represents the destination device and file specification, and *input-filespec* represents the source device and file specification.

2. After you have entered the last command string, the system responds with the following prompt:

```
Mount input volume in <device>;  Continue?
```

   Type a Y followed by a carriage return when you have mounted the input volume:

3. The system then prints the next instruction for you to mount the output volume:

```
Mount output volume in <device>;  Continue?
```

   Type a Y followed by a carriage return in response to the last message after you have mounted the output volume:

4. Unlike the single-volume transfer, the double-volume transfer involves only one cycle of mounting the input and output volumes. When the file transfer is complete, PIP prints the following instruction:

```
Mount system volume in <device>;  Continue?
```

   When you mount your system volume and type a Y followed by a carriage return in response to the last instruction, you terminate the copy operation.

CREATE

The CREATE command creates or extends a file with a specific name, location, and size on the block replaceable volume that you specify.

In the command syntax illustrated above, *filespec* represents the device and file specifications of the file you wish to create or extend. If you are using the CREATE command to create a file, this command only creates a directory entry for the file. This command does not store any data in a file. You must specify both the file name and type of the file you wish to create or extend.

If you type a carriage return after typing CREATE, the system prompts *File?*.

The following sections describe the options you can use with the CREATE command.

/ALLOCATE:n Use this option following the file specification to allocate *n* blocks for the file you are creating, where *n* represents a decimal number. A value of -1 for *n* indicates a file of the maximum size available on the volume. If you do not use /ALLOCATE, the system assumes one block.

/EXTENSION:n Use this option to extend an existing file you specify by *n* blocks, where *n* is a decimal number. When you use this option following the file specification, make sure that there is enough unused space on the volume for the size you specify (use the DIRECTORY/FULL command to do this). If you do not supply a value with /EXTENSION, the system assumes one block.

The following example illustrates the procedure for extending a file with the CREATE command. In this example, BUILD.MAC is extended by 20 blocks. First, a DIRECTORY/FULL command determines whether there is available space adjacent to BUILD.MAC.

```
.DIRECTORY/FULL DX0:
 05-DEC-79
MYPROG.MAC     36P 19-NOV-79        TM     .MAC     25  27-NOV-79
VTMAC .MAC      7  19-NOV-79        SYSMAC.MAC     41  19-NOV-79
< UNUSED >     25                   RT11SJ.SYS     67  19-NOV-79
TT     .SYS     2  19-NOV-79        DX     .SYS     3  19-NOV-79
LELA   .LBM     1  05-DEC-79        BUILD  .MAC    80  19-NOV-79
< UNUSED >    199
 9 Files, 262 Blocks
 224 Free blocks
```

Next the CREATE command extends BUILD.MAC by 20 blocks.

```
.CREATE DX0:BUILD.MAC/EXTENSION:20
```

/START:n Use this option to specify the starting block number of the file you are creating. The argument *n* represents a decimal block number. If you do not use /START, the system uses the first available space on the volume.

The following example illustrates the procedure for creating a file with the CREATE command. In this example, SWAP.SYS is restored after having been previously deleted. First, a DIRECTORY/DELETED command establishes the starting block numbers of the deleted files on DX0:

```
.DIRECTORY/DELETED DX0:
 05-DEC-79
SWAP  .SYS     25  19-NOV-79   117   EMPTY.FIL   179  31-OCT-79   315
 0 Files, 0 Blocks
 204 Free blocks
```

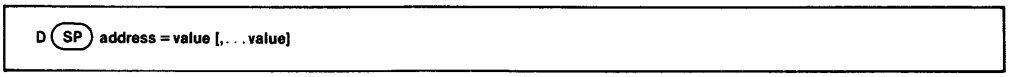
Next, the CREATE command restores SWAP.SYS, starting at block 117, and using the /ALLOCATE:n option to allocate 25 blocks.

```
.CREATE DX0:SWAP.SYS/START:117/ALLOCATE:25
```

See the *RT–11 Software Support Manual* for a detailed description of the RT–11 file structure.

D

The D (Deposit) command deposits values in memory, beginning at the location you specify.



In the command syntax illustrated above, *address* represents an octal address that, when added to the relocation base value from the Base command (if you used one), provides the actual address where the system must deposit the values. The argument *value* represents the new contents of the address. If you do not specify a value, the system assumes a value of 0. If you specify more than one value and separate the values by commas, the system deposits the values in sequential locations, beginning at the location you specify.

The Deposit command accepts both word and byte addresses, but it always executes the command as though you specified a word address. (If you specify an odd address, the system decreases it by one to make it even.) The Deposit command stores all values as word quantities.

Use commas to separate multiple values in the command line. Two or more adjacent commas cause the system to deposit zeroes at the location you specify and at the following locations, if indicated.

Note that you cannot specify an address that references a location outside the area of the background job. You can use the D command with GET and START to temporarily alter a program's execution. Use the SAVE command before START to make the alteration permanent.

The following command deposits zeroes into locations 300, 302, 304, and 306.

```
.D 300=,,,
```

The next command sets the base address to 0.

```
.B
```

The following command deposits 3705 into location 1000.

```
.D 1000=3705
```

The next command sets the relocation base to 1000.

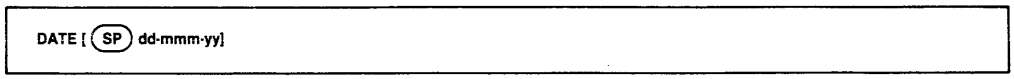
```
.B 1000
```

The next command puts 2503 into location 1500 (offset of 500 from the last B command) and 22 into location 1502.

```
.D 500=2503,22
```

DATE

Use the DATE command to set or to inspect the current system date.



In the command syntax shown above, *dd* represents the day (a decimal number from 1 to 31); *mmm* represents the first three characters of the name of the month; and *yy* represents the year (a decimal number from 73 to 99).

To enter a date into the system, as soon as you bootstrap the system specify the date in the format described above. The system uses this date for newly created files, for files that you transfer to magtape or cassette, and for listing files. The following example enters the current date.

```
.DATE 18-MAY-77
```

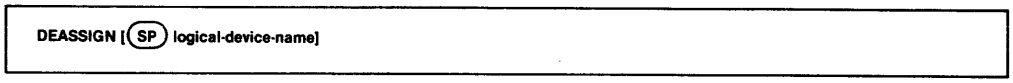
To display the current system date, type the DATE command without an argument, as this example shows.

```
.DATE
18-May-77
```

The FB and XM monitors automatically increment the date at midnight each day. The SJ monitor increments the date only if you select timer support as a system generation special feature. Note that you can also select automatic end-of-month date advancement through system generation.

DEASSIGN

The DEASSIGN command disassociates a logical device name from a physical device name.



In the command syntax illustrated above, *logical-device-name* represents an alphanumeric name, from one to three characters long, that is assigned to a particular device. Note that spaces and tabs are not permitted in the logical device name.

To remove the assignment of a particular logical device name to a physical device, specify that logical device name in the command line. The following example disassociates the logical name INP: from the physical device to which it is assigned.

```
.DEASSIGN INP:
```

If you specify a logical name that is not currently assigned, the system prints an error message, as this example shows.

```
.DEASSIGN INP:
?KMON-F-Logical name not found
```

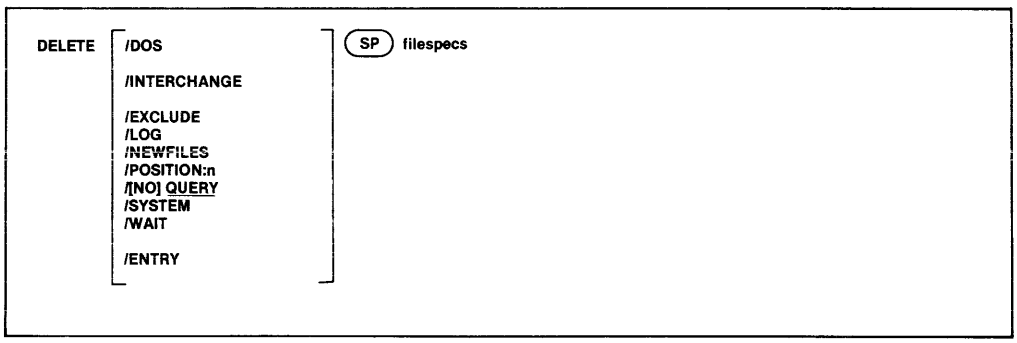
To disassociate all logical names from physical devices, type the DEASSIGN command without an argument. The following example disassociates all logical device names (except SY:) from physical devices and resets the logical names DK: and SY: to represent the system volume.

```
.DEASSIGN
```

If DK: is assigned to a non-system device (such as DX1:, for example), the following command disassociates DK: from DX1: and restores the default association of DK: to SY:, the system device.

```
.DEASSIGN DK:
```

The DELETE command deletes the files you specify.



In the command syntax shown above, *filespecs* represents the files to be deleted. You can specify up to six files; separate them with commas. You can enter the DELETE command as one line, or you can rely on the system to prompt you for information. If you omit the file specification, the DELETE command prompts *Files?*. If you delete a file accidentally, it may be possible to recover the file if you act immediately (see CREATE). A procedure for doing this is described in Chapter 8.

The system has a special way of handling system (.SYS) files and files that cover bad blocks (.BAD files). So that you do not delete system files by accident when you use a wildcard in the file specification, the system requires you to use the /SYSTEM option when you need to delete system files. To delete a .BAD file, you must specify it by explicitly giving its file name and file type. Since .BAD files cover bad blocks on a device, you do not need to copy, delete, or otherwise manipulate these files. To delete a protected file (a "P" next to the block size of a file's directory entry denotes protection) use the RENAME/NOPROTECTION command.

Another feature of the DELETE command is that the system unless using /LOG or /NOQUERY requests confirmation from you before it deletes a file. You must respond to the query message by typing Y followed by a carriage return in order to execute the command.

The following sections describe the options you can use with the DELETE command.

/DOS Use this option to delete a file that is in DOS–11 or RSTS/E format. The valid devices for this type of file are disks or DECtapes. You cannot use any other option in combination with /DOS.

/ENTRY Use this option to delete a job from the queue. Use /ENTRY when QUEUE is running as a foreground or system job (see Chapter 20, Queue Package).

When you use /ENTRY, you do not have to specify the input files in the job, only the job name. If you have not specified a job name, the system uses the first file name in the job as the job name. The following example deletes MILLER from the queue:

```
.DELETE/ENTRY MILLER
```

If QUEUE is printing a job when you delete that job, QUEUE immediately stops processing that job.

/EXCLUDE This option deletes all the files on a device except the ones you specify. The following command, for example, deletes all files from DX0: except .SAV files.

```
.DELETE/EXCLUDE DX0:*.SAV
?PIP-W-No .SYS action
 Files deleted:
DX0:ABC.OLD    ? Y
DX0:AAF.OLD    ? Y
DX0:COMB.      ? Y
DX0:MERGE.OLD  ? Y
```

/INTERCHANGE Use this option to delete from a diskette a file that is in interchange format. You cannot use any other option with /INTERCHANGE.

/LOG This option lists on the terminal a log of the files that are deleted by the current command. Note that if you specify /LOG, the system does not ask you for confirmation before execution proceeds (that is, /LOG implies /NOQUERY). Use both /LOG and /QUERY to invoke logging and querying.

/NEWFILES Use this option to delete only the files that have the current system date. This is a convenient way to remove all the files that you just created in a session at the computer. The following example deletes the files created today.

```
.DELETE/NEWFILES DX1:*.BAK
 Files deleted:
DX1:MERGE.BAK ? Y
```

/POSITION[:n] You can use this option when you delete files from cassette. It permits you to move the tape and perform an operation at the point you specify. Omitting the argument *n* has the same effect as setting *n* equal to 0 (*n* is interpreted as a decimal number). The /POSITION:n option has the following effect:

1. If *n* is 0:
   The cassette rewinds and the system searches for the file you specify. If you specify more than one file, or if you use a wildcard in the file specification, the cassette rewinds before each search.

2. If *n* is a positive integer:
   The system starts from the cassette's present position and searches for the file you specify. If the system does not find the file you specify before it reaches the *n*th file from its starting position, it deletes the *n*th file.

Note that if the starting position is not the beginning of the tape, it is possible that the system will not find the file you specify, even though it does exist on the tape.

3. If *n* is a negative integer:
   The cassette rewinds, then the system follows the procedure outlined in step 2 above.

/QUERY Use this option to request a confirmation message from the system before it deletes each file. This option is particularly useful on operations that involve wildcards, when you may not be completely sure which files the system selected for the operation. This is the default mode of operation. Note that specifying /LOG eliminates the automatic query; you must specify /QUERY with /LOG to retain the query function. You must respond to a query message by typing Y (or anything that begins with a Y) and a carriage return to initiate execution of a particular operation. The system interprets any other response as NO; it does not perform the operation. The following example shows querying. Only one file is deleted.

```
.DELETE/QUERY DX1:*.*
 Files deleted:
DX1:ABC.MAC    ? N
DX1:AAF.MAC    ? Y
DX1:MERGE.FOR  ? N
```

/NOQUERY This option suppresses the confirmation message the system prints before it deletes each file.

/SYSTEM Use this option if you need to delete system (.SYS) files. If you omit this option, the system files are excluded from the DELETE operation, and a message is printed on the terminal. (Note that the system prints this message only when system files might otherwise be included in the operation.)

/WAIT This option is useful if you have a single-disk system. When you use this option, the system initiates the DELETE operation but then pauses for you to mount the volume on which you want the operation to take place. When the system pauses, it prints *Mount input volume in <device>; Continue?*, where *<device>* represents the device into which you mount the volume. When the volume is mounted, type Y followed by a carriage return.

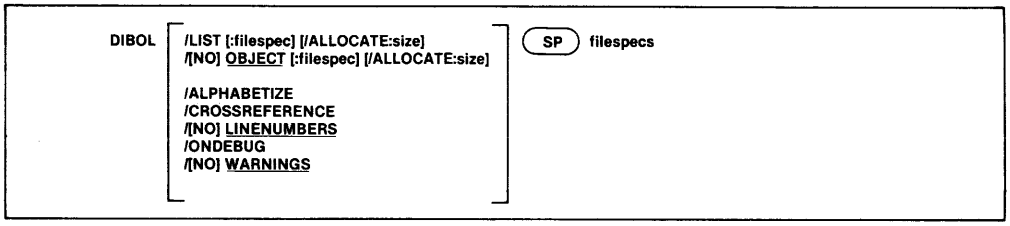
The following example deletes FILE.MAC from an RK05 disk:

```
.DELETE/WAIT RK0:FILE.MAC
Mount input volume in RK0:; Continue? Y
RK0:FILE.MAC? Y
Mount system volume in RK0:; Continue? Y
```

This option is invalid with /INTERCHANGE and /DOS.

DIBOL

The DIBOL command invokes the DIBOL compiler to compile one or more source programs.



In the command syntax illustrated above, *filespecs* represents one or more files to be included in the compilation. If you omit a file type for an input file, the system assumes .DBL. Output default file types are .LST for listing files and .OBJ for object files. To compile multiple source files into a single object file, separate the files by plus (+) signs in the command line. Unless you specify otherwise, the system creates an object file with the same name as the first input file and gives it an .OBJ file type. To compile multiple files in independent compilations, separate the files by commas (,) in the command line. This generates a corresponding object file for each set of input files.

Language options are position-dependent — that is, they have different meanings depending on where you place them in the command line. Options that qualify a command name apply across the entire command string. Options that follow a file specification apply only to the file (or group of files separated by plus signs) they follow in the command string.

You can enter the DIBOL command as one line, or you can rely on the system to prompt you for information. The DIBOL command prompt is *Files?* for the input specification.

The *DIBOL–11 Language Reference Manual* contains more detailed information about using DIBOL. The following sections describe the options you can use with the DIBOL command.

/ALLOCATE:size Use this option with /LIST or /OBJECT to reserve space on the device for the output file. The argument *size* represents the number of blocks of space to allocate. The meaningful range for this value is from 1 to 32767. A value of –1 is a special case that creates the largest file possible on the device.

/ALPHABETIZE Use this option to alphabetize entries in the symbol and label tables. This is useful for program maintenance and debugging.

/CROSSREFERENCE This option generates a symbol cross-reference section in the listing to which it adds as many as four separate sections to the listing. These sections are: (1) symbol cross-reference table, (2) label cross-reference table, (3) external subroutine cross-reference table, (4) COMMON cross-reference table. Note that the system does not generate a

listing by default. You must also specify /LIST in the command line to get a cross-reference listing.

/LINENUMBERS This option generates line numbers for the program during compilation. These line numbers are referenced by the symbol table segment, label table segment, and the cross-reference listing; they are especially useful in debugging DIBOL programs. This is the default operation.

/NOLINENUMBERS This option suppresses the generation of line numbers during compilation, which produces a smaller program and optimizes execution speed. Use this option to compile only programs that are already debugged; otherwise the DIBOL error messages are difficult to interpret.

/LIST[:filespec] You must specify this option to produce a DIBOL compilation listing. The /LIST option has different meanings depending on where you place it in the command line.

The /LIST option produces a listing on the line printer when /LIST follows the command name. For example, the following command line produces a line printer listing after compiling a DIBOL source file:

```
.DIBOL/LIST MYPROG<RET>
```

When the /LIST option follows the file specification, it produces a listing file. For example, the following command line produces the listing file DK:MYPROG.LST after compiling a DIBOL source file:

```
.DIBOL MYPROG/LIST<RET>
```

If you specify /LIST in the list of options that immediately follows the command name, but omit a file specification, the DIBOL compiler generates a listing that prints on the line printer. If you follow /LIST with a device name, the system creates a listing file on that device. If the device is a file-structured device, the system stores the listing file on that device, assigning it the same name as the input file with a .LST file type. The following command produces a listing on the terminal.

```
.DIBOL/LIST:TT: A
```

The next command creates on RK3: a listing file called A.LST.

```
.DIBOL/LIST:RK3: A
```

If the /LIST option contains a name and file type to override the default of .LST, the system generates a listing file with that name. The following command, for example, compiles A.DBL and B.DBL together, producing on device DK: files A.OBJ and FILE1.OUT:

```
.DIBOL/LIST:FILE1.OUT A+B
```

Another way to specify /LIST is to type it after the file specification to which it applies. To produce a listing file with the same name as a particular input file, you can use a command similar to this one:

```
.DIBOL A+B/LIST:RK3:
```

The command shown above compiles A.DBL and B.DBL together, producing files DK:A.OBJ and RK3:B.LST. If you specify a file name on a /LIST option following a file specification in the command line, it has the same meaning as when it follows the command. The following two commands have the same results:

```
.DIBOL A/LIST:B

.DIBOL/LIST:B A
```

Both commands generate as output files A.OBJ and B.LST.

Remember that file options apply only to the file (or group of files that are separated by plus signs) they follow in the command string. For example:

```
.DIBOL A/LIST,B
```

This command compiles A.DBL, producing A.OBJ and A.LST. It also compiles B.DBL, producing B.OBJ. However, it does not produce any listing file for the compilation of B.DBL.

/OBJECT[:filespec] Use this option to specify a file name or device for the object file. Because DIBOL creates object files by default, the following two commands have the same meaning:

```
.DIBOL A

.DIBOL/OBJECT A
```

Both commands compile A.DBL and produce A.OBJ as output. The /OBJECT option functions like the /LIST option; it can be either a command or a file qualifier.

As a command option, /OBJECT applies across the entire command string. The following command, for example, compiles A.DBL and B.DBL separately, creating object files A.OBJ and B.OBJ on RK1:.

```
.DIBOL/OBJECT:RK1: A,B
```

Use /OBJECT as a file option to create an object file with a specific name or destination. The following command compiles A.DBL and B.DBL together, creating files B.LST and B.OBJ.

```
.DIBOL A+B/LIST/OBJECT
```

/NOOBJECT Use this option to suppress creation of an object file. As a command option, /NOOBJECT suppresses all object files; as a file option, it suppresses only the object file produced by the related input files. In this command, for example, the system compiles A.DBL and B.DBL together, producing files A.OBJ and B.LST. It also compiles C.DBL and produces C.LST, but does not produce C.OBJ.

```
.DIBOL A+B/LIST, C/NOOBJECT/LIST
```

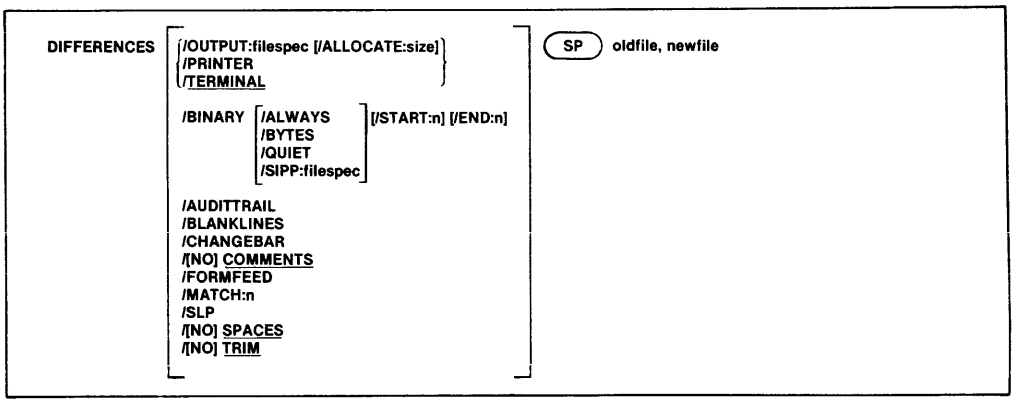
/ONDEBUG This option includes a symbol table in the object file. You can then use a debugging program to find and correct errors in the object file.

/WARNINGS Use this option to include warning messages in DIBOL compiler diagnostic error messages. These messages call certain conditions to your attention, but they do not interfere with the compilation. This is the default operation.

/NOWARNINGS Use this option to suppress warning messages during compilation.

DIFFERENCES

The DIFFERENCES command compares two files and lists the differences between them.



In the command syntax shown above, *oldfile* represents the first file to be compared and *newfile* represents the second. The default output device is the console terminal. The default file type for input files is .MAC; for output files it is .DIF. You can specify the entire command on one line, or you can rely on the system to prompt you for information. The DIFFERENCES command prompts are *File 1?* and *File 2?*.

The DIFFERENCES command is particularly useful when you want to compare two similar versions of a source or binary program, typically, an updated version against a backup version. A file comparison listing highlights the changes made to a program during an editing session. The following sections describe the various options you can use with the DIFFERENCES command. Following the descriptions of the options is a sample listing and an explanation of how to interpret it.

The DIFFERENCES command is also useful for creating command files that can install patches to backup versions of programs so they match the updated versions. The /SLP and /SIPP;filespec options are designed especially for this purpose.

/ALLOCATE:size Use this option with /OUTPUT and /SIPP to reserve space on the device for the output listing file. The argument *size* represents the number of blocks of space to allocate. The meaningful range for this value is from 1 to 32767. A value of -1 is a special case that creates the largest file possible on the device.

/ALWAYS When you use this option with /BINARY or /SIPP:filespec, the system creates an output file regardless of whether there are any differences between the two input files. This option is useful when running BATCH streams to prevent job step failures due to the absence of a DIFFERENCES output file.

The /ALWAYS option is position dependent. That is, you must use it immediately after the output file to which you want it to apply. If you use it at the end of the command string, it applies to all output files.

/AUDITTRAIL Use this option with /SLP to specify an audit trail. The /SLP option, described below, creates a command file which, when run with the source language patch program (SLP), can patch *oldfile* so it matches *newfile*. When you use SLP to modify a file, it creates an output file that has audit trails. An audit trail is a string of characters that appears in the right margin of each line that has been changed by the modification procedure. The audit trail keeps track of the patches you make to the patched source file.

By default, SLP uses the following characters for the audit trail:

```
;**NEW**
```

When you use the /AUDITTRAIL option, the system prints the following prompt at the terminal.

```
Audit trail?
```

Enter a string of up to 12 ASCII characters that you want to use in place of the default audit trail. Do not use the slash (/) in the audit trail.

/BINARY When you use this option, the system compares two binary files and lists the differences between them. This option is useful for comparing and relocatable image files (that is, machine runnable programs and object files) and provides a quick way of telling whether two files are identical. For example, you can use /BINARY to tell whether two versions of a program produce identical output.

When you use /BINARY and do not specify an output file, the system prints output at the terminal according to the following general syntax:

bbbbbb ooo/ ffffff ssssss xxxxxx

where:

bbbbbb	represents the octal block number of the block that contains the difference
ooo	represents the octal offset within the block that contains the difference
ffffff	represents the value in the first file you are comparing
ssssss	represents the value in the second file you are comparing
xxxxxx	represents the logical exclusive OR of the two values in the input files

If you use the /OUTPUT:filespec option with /BINARY, the system stores the differences listing in the file you specify (if there are any differences found), instead of printing the differences at the terminal.

/BLANKLINES Use this option to include blank lines in the file comparison. Normally, the system disregards blank lines.

/BYTES When you use this option with /BINARY, the system lists the differences byte-by-byte.

/CHANGEBAR Use this option to create an output file that contains *newfile* with a changebar character next to the lines in *newfile* that differ from *oldfile*. The system inserts a vertical bar next to each line that has been added to *newfile*, and a bullet (lower-case letter o) next to each line that has been deleted.

The output defaults to the terminal. Use the /PRINTER option to list the output to the line printer. Specify an output file with the /OUTPUT:filespec option.

The sample that follows creates a listing of RTLIB.MAC with a changebar character at the left margin of each line that is different from RTLIB.BAK:

```
DIFFERENCES/CHANGEBAR RTLIB.BAK,RTLIB.MAC
```

/COMMENTS When you use this option, the system includes in the file comparison all assembly language comments it finds in the two files. (Comments are preceded by a semicolon on the same line.) This is the default operation.

/NOCOMMENTS Use this option to exclude comments from the comparison. (Comments are preceded by a semicolon on the same line.) This is useful if you are comparing two MACRO source programs with similar content but different format.

/END[:n] Use this option with /BINARY to specify the ending block number of the file comparison, where *n* is an octal number that represents the ending block number. If you do not supply a value with /END, the system defaults to the last block of the file or volume.

/FORMFEED Use this option to include form feeds in the output listing. Normally, the system compares form feeds but does not include them in the output listing.

/MATCH[:n] Use this option to specify the number of lines from each file that must agree to constitute a match. The value *n* is an integer in the range 1-200. The default value for *n* is 3.

/OUTPUT:filespec Use this option to specify a device and file name for the output listing file. Normally, the listing appears on the console terminal. If you omit the file type for the listing file, the system uses .DIF. Note that the system creates this file only if there are any differences found. Use the /ALWAYS option if you want the system to create an output file regardless of whether any differences are found.

/PRINTER Use this option to print a listing of differences on the printer. Normally, the listing appears on the console terminal.

/QUIET When you use this option with /BINARY, the system suppresses printing the differences at the terminal and prints *?BINCOM-W-Files are different,* if applicable.

/SIPP:filespec Use this option with /BINARY to output a file that you can use as an input command file to the save image patch program SIPP, where *filespec* represents the name of the output file.

The file you create with /SIPP can patch *oldfile* so it matches *newfile.*

The example that follows creates an input command file which, when run with SIPP, patches DEMOF1.BAK so it matches DEMOF1.SAV.

```
DIFFERENCES/BINARY/SIPP:PATCH.COM  DEMOF1.BAK, DEMOF1.SAV
```

To execute the input command file created by /SIPP, see Chapter 22, Save Image Patch Program (SIPP).

/SLP Use this option with /OUTPUT:filespec to create a command file that, when run with the source language patch utility SLP, patches *oldfile* to match *newfile.* If you do not use the /OUTPUT:filespec option with /SLP, the system prints the command file at the terminal.

The sample that follows creates the command file PATCH.COM. PATCH.COM can be used as input to the program SLP to patch RTLIB.BAK so that it matches RTLIB.MAC.

```
.DIFFERENCES/SLP/OUTPUT:PATCH.COM RTLIB.BAK,RTLIB.MAC
```

To execute the command file you create with /SLP, see Chapter 24, Source Language Patch Program (SLP).

/SPACES This option includes spaces and tabs in the file comparison. This is the default operation and is particularly useful when you are comparing two text files and must pay careful attention to spacing.

/NOSPACES Use this option to exclude spaces and tabs from the file comparison. This is useful when you are comparing two source programs with similar contents but different formats.

/START[:n] Use this option with /BINARY to specify the starting block number of the file comparison, where *n* represents the octal starting block number. If you do not supply a value with /START, the system defaults to the first block in the file.

/TERMINAL Use this option to cause the list of differences to appear on the console terminal. This is the default operation.

To understand how to interpret the output listing, first look at the following two text files.

```
.TYPE FILE1.TXT
HERE'S A BOTTLE AND AN HONEST FRIEND!
  WHAT WAD YE WISH FOR MAIR, MAN?
WHA KENS, BEFORE HIS LIFE MAY END,
  WHAT HIS SHAME MAY BE O' CARE, MAN?
```

```
THEN CATCH THE MOMENTS AS THEY FLY,
  AND USE THEM AS YE OUGHT, MAN: --
BELIEVE ME, HAPPINESS IS SLY,
  AND COMES NOT AY WHEN SOUGHT, MAN.

                        --SCOTTISH SONG


.TYPE FILE2.TXT
HERE'S A BOTTLE AND AN HONEST FRIEND!
  WHAT WAD YE WISH FOR MAIR, MAN?
WHA KENS, BEFORE HIS LIFE MAY END,
  WHAT HIS SHARE MAY BE O' CARE, MAN?
THEN CATCH THE MOMENTS AS THEY FLY,
  AND USE THEM AS YE OUGHT, MAN: --
BELIEVE ME, HAPPINESS IS SHY,
  AND COMES NOT AY WHEN SOUGHT, MAN.

                        --SCOTTISH SONG
```

Notice that in the fourth line of FILE1.TXT, *shame* should be *share*; in the seventh line, *sly* should be *shy*.

The following command compares the two files, creating a listing file called DIFF.TXT.

```
.DIFFERENCES/MATCH:1/OUTPUT:DIFF.TXT FILE1.TXT,FILE2.TXT
?SRCCOM-W-Files are different
```

The following listing shows file DIFF.TXT.

```
.TYPE DIFF.TXT
1) DK:FILE1.TXT
2) DK:FILE2.TXT
**********
1)1       WHAT HIS SHAME MAY BE O' CARE, MAN?
1)      THEN CATCH THE MOMENTS AS THEY FLY,
****
2)1       WHAT HIS SHARE MAY BE O' CARE, MAN?
2)      THEN CATCH THE MOMENTS AS THEY FLY,
**********
1)1     BELIEVE ME, HAPPINESS IS SLY,
1)        AND COMES NOT AY WHEN SOUGHT, MAN.
****
2)1     BELIEVE ME, HAPPINESS IS SHY,
2)        AND COMES NOT AY WHEN SOUGHT, MAN.
**********
```

If the files are different, the system always prints the file name of each file as identification:

```
1) DK:FILE1.TXT
2) DK:FILE2.TXT
```

The numbers at the left margin have the form *n)m*, where *n* represents the source file (either 1 or 2) and *m* represents the page of that file on which the specific line is located.

The system next prints ten asterisks and then lists the differences between the two files. The /MATCH:n option was used in this example to set to 1 the number of lines that must agree to constitute a match.

The first three lines of the song are the same in both files, so they do not appear in the listing. The fourth line contains the first discrepancy. The system prints the fourth line from the first file, followed by the next matching line as a reference.

```
1)1         WHAT HIS SHAME MAY BE O' CARE, MAN?
1)        THEN CATCH THE MOMENTS AS THEY FLY,
****
```

The four asterisks terminate the differences section from the first file.

The system then prints the fourth line from the second file, again followed by the next matching line as a reference:

```
2)1         WHAT HIS SHARE MAY BE O' CARE, MAN?
2)        THEN CATCH THE MOMENTS AS THEY FLY,
**********
```

The ten asterisks terminate the listing for a particular difference section.

The system scans the remaining lines in the files in the same manner. When it reaches the end of each file, it prints *?SRCCOM-W-Files are different* on the terminal.

If you compare two files that are identical, the system does not create an output listing, but prints:

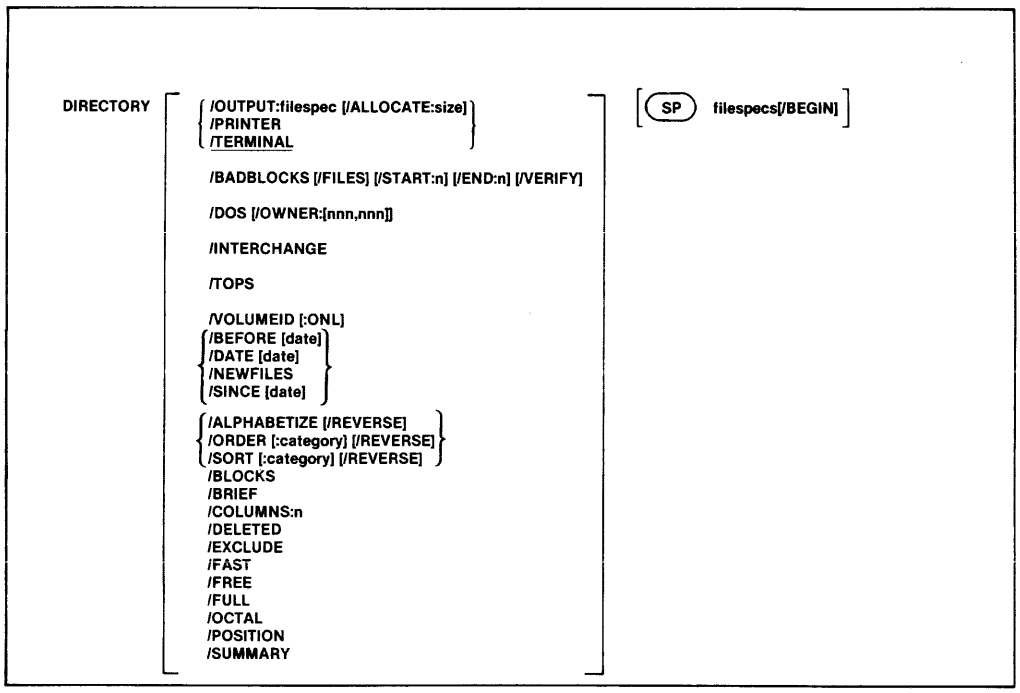
```
?SRCCOM-I-No differences found
```

/TRIM Use the /TRIM option with /SLP to ignore tabs and spaces that appear at the ends of source lines. This is the default setting.

/NOTRIM Use /NOTRIM with /SLP to include in the comparison spaces and tabs that appear at the ends of source lines. /TRIM is the default setting.

DIRECTORY

The DIRECTORY command lists information you request about a device, a file, or a group of files.



In the command syntax shown above, *filespecs* represents the device, file, or group of files whose directory information you request. The DIRECTORY command can list directory information about a specific device, such as the number of files stored on the device, their names, and their creation dates. It can list details about certain files including their names, their file types, and their size in blocks. You can specify up to six files explicitly, but you can obtain directory information about many files by using wildcards in the file specification. The DIRECTORY command can also print a device directory summary, organized in several ways, such as alphabetical or chronological.

Normally, the DIRECTORY command prints listings in two columns on the terminal. Read these listings as you would read a book; read across the columns, moving from left to right, one row at a time. Directory listings that are sorted (with /ALPHABETIZE, /ORDER, or /SORT) are an exception to this. Read these listings as you would a telephone directory, by reading the left column from top to bottom, then reading the right column from top to bottom.

The DIRECTORY command does not prompt you for any information. If you omit the file specification, the system lists directory information about device DK:, as this example shows.

```
.DIRECTORY
 27-Nov-79
RT11SJ.SYS     67P 03-Jul-79       RT11FB.SYS     80P 13-Aug-79
RT11BL.SYS     63P 15-Mar-79       DX     .SYS     3P 13-Aug-79
SWAP   .SYS    25P 13-Aug-79       TT     .SYS     2P 13-Aug-79
DP     .SYS     3P 13-Aug-79       DY     .SYS     4P 13-Aug-79
LP     .SYS     2P 27-Nov-79       PIP    .SAV    16  25-Jul-79
DUP    .SAV    41  26-Mar-79       RESORC.SAV    15  13-Aug-79
EDIT   .SAV    19  13-Aug-79       STARTS.COM     1  27-Aug-79
SIPP   .SAV    14  13-Aug-79
 15 Files, 413 Blocks
 73 Free blocks
```

A "P" next to the block size number of a file's directory entry indicates that the file is protected from deletion (see RENAME/PROTECTION).

If you specify only a device in the file specification, the system lists directory information about all the files on that device. If you specify a file name, the system lists information about just that file, as this example shows.

```
.DIRECTORY DX0:RT11FB.SYS
 10-Dec-79
RT11FB.SYS     80P 13-Aug-79
 1 File, 80 Blocks
 4 Free blocks
```

The following sections describe the options you can use with the DIRECTORY command and provide sample directory listings. Some of the options accept a date or part of a date as an argument. The syntax for specifying the date is:

[:dd][:mmm][:yy]

where:

dd	represents the day (a decimal integer in the range 1–31)
mmm	represents the first three characters of the name of the month
yy	represents the year (a decimal integer in the range 73–99)

The default value for the date is the current system date. If you specify just the day, the system interprets it as the given day of the current month and year. If you specify just the month, the system interprets it to be the first day of the given month in the current year. If you specify only the year, the system interprets it as the start of that year. If the current system date is not set, it is considered 0 (the same as for an undated file in a directory listing).

If you have selected timer support through the system generation process, but have not selected automatic end-of-month date advancement, make sure that you set the date of the beginning of each month with the DATE command. If you fail to set the date at the beginning of each month, the system prints *-BAD-* in the creation date column of each file created beyond the end-

of-month. (Note that you can eliminate *-BAD-* by using the RENAME/SETDATE command after you set the date.)

/ALLOCATE:size Use this option with /OUTPUT to reserve space on the device for the output listing file. The argument *size* represents the number of blocks of space to allocate. The meaningful range for this value is from 1 to 32767. A value of -1 is a special case that creates the largest file possible on the device.

/ALPHABETIZE This option lists the directory of the device you specify in alphabetical order by file name and file type. It has the same effect as the /ORDER:NAME option. Note that this option sorts numbers after letters.

/BADBLOCKS Sometimes devices (disks and DECtapes) have bad blocks, or they develop bad blocks as a result of use and age. Use the /BADBLOCKS option to scan a device and locate bad blocks on it. The system prints the absolute block number of these blocks on the devices that return hardware errors when the system tries to read them. This procedure does not destroy data that is already stored on the device. Remember that block numbers are listed in both octal and decimal and the first block on a device is block 0. If a device has no bad blocks, an informational message prints on the terminal.

```
.DIRECTORY/BADBLOCKS DX1:
?DUP-I-No bad blocks detected
```

If /BADBLOCKS is the only option in the command line, the volume being scanned does not need a valid RT–11 directory structure.

/BEFORE[date] This option prints a directory of files created before the date you specify. The following command lists on the terminal all files stored on device DX1: that were created before December 1979.

```
.DIRECTORY/BEFORE:DEC DX1:
 14-Dec-79
MYPROG.MAC     36P 19-Nov-79       TM     .MAC     25  27-Nov-79
VTMAC .MAC      7  19-Nov-79       SYSMAC.MAC     41  19-Nov-79
RT11SJ.SYS      0  19-Nov-79       RT11SJ.SYS     67  19-Nov-79
TT     .SYS     2  19-Nov-79       DX     .SYS     3  19-Nov-79
BUILD .MAC    100  19-Nov-79
 9 Files, 281 Blocks
 180 Free blocks
```

/BEGIN This option lists the directory of the device you specify, beginning with the file you name and including all the files that follow it in the directory. The occurrence of file names in the listing is the same as the order of the files on the device.

The following example lists the file VTMAC.MAC on device DX0: and all the files that follow it in the directory.

```
.DIRECTORY DX0:VTMAC.MAC/BEGIN
 10-Dec-79
VTMAC .MAC     15  10-Aug-79       DIR    .SAV     17  03-Aug-79
RK    .SYS      3  13-Aug-79       EDIT   .SAV     19  03-Aug-79
STARTS.COM      1  27-Aug-79       DD     .SYS      5  19-Aug-79
SRCCOM.SAV     13  13-Aug-79       BINCOM.SAV      11  05-Oct-79
SLP   .SAV      9  13-Aug-79       SIPP   .SAV     14  05-Oct-79
 10 Files, 107 Blocks
 73 Free blocks
```

/BLOCKS This option prints a directory of the device you specify and includes the starting block number in decimal (or in octal if you use /OCTAL) of all the files listed. The following example lists the directory of DX0:, including the starting block numbers of files.

```
.DIRECTORY/BLOCKS DX0:
 14-Dec-79
FSM   .MAC     31P 19-Nov-79  2955   BATCH .MAC   102P 19-Nov-79  2986
ELCOPY.MAC      8P 19-Nov-79  3088   ELINIT.MAC    15P 19-Nov-79  3096
ELTASK.MAC     15P 19-Nov-79  3111   ERROUT.MAC    48P 19-Nov-79  3126
ERRTXT.MAC      9P 19-Nov-79  3174   SYCND .BL      3P 19-Nov-79  3183
SYSTBL.BL       4P 19-Nov-79  3186   SYCND .DIS     5P 19-Nov-79  3190
SYSTBL.DIS      4P 19-Nov-79  3195   SYCND .HD      5P 19-Nov-79  3199
ABSLOD.SAV     48  15-Mar-76  3204   CHESS .SAV    40  17-Aug-75  3252
PETAL .SAV     36  11-Sep-75  3292   LAMP  .SAV    29  16-Mar-79  3328
WUMPUS.SAV     30  16-Mar-79  3357
 17 Files, 348 Blocks
 138 Free blocks
```

/BRIEF This option lists only file names and file types, omitting file lengths and associated dates. It produces a five-column listing, as the following example shows.

```
.DIRECTORY/BRIEF RK1:
 14-Dec-79
SWAP   .SYS     RT11SJ.SYS     RT11FB.SYS     RT11BL.SYS     TT     .SYS
DT     .SYS     DP     .SYS    DX     .SYS    DY     .SYS    RF     .SYS
RK     .SYS     DL     .SYS    DM     .SYS    DS     .SYS    DD     .SYS
LP     .SYS     LS     .SYS    CR     .SYS    MS     .SYS    MTHD   .SYS
DISMT1.COM      MMHD   .SYS    NUMBER.PAS     WLOCK  .SAV    MYPROG.MAC
PROG   .MAP     ANTONY.BAK     MSHD   .SYS     NL     .SYS    PC     .SYS
PD     .SYS     CT     .SYS    BA     .SYS    MYPROG.SAV     ODT    .SAV
 35 Files, 408 Blocks
 78 Free blocks
```

/COLUMNS:n Use this option to list a directory in a specific number of columns. The value *n* represents an integer in the range 1-9. Normally, the system uses two columns for regular listings and five columns for brief listings. The following example lists the directory information for device DX1: in one column.

```
.DIRECTORY/COLUMNS:1 DX1:
 29-Nov-79
SWAP   .SYS     25P 19-Nov-79
RT11SJ.SYS      67P 19-Nov-79
RT11FB.SYS      80P 19-Nov-79
RT11BL.SYS      64P 19-Nov-79
TT     .SYS      2P 19-Nov-79
DT     .SYS      3P 19-Nov-79
DP     .SYS      3P 19-Nov-79
 7 Files, 244 Blocks
 242 Free blocks
```

/DATE[date] Use this option to include in the directory listing only those files with a certain date. The following command lists all the files on device DX0: that were created on 13 August 1979.

```
.DIRECTORY/DATE:13:AUG:79 DX0:
 15-Sep-79
RT11SJ.SYS      67P 13-Aug-79       RT11FB.SYS      80P 13-Aug-79
RT11BL.SYS      63P 13-Aug-79       DX     .SYS      3P 13-Aug-79
SWAP   .SYS     25P 13-Aug-79       TT     .SYS      2P 13-Aug-79
DP     .SYS      3P 13-Aug-79       DY     .SYS      4P 13-Aug-79
LP     .SYS      2P 13-Aug-79       PIP    .SAV     16  13-Aug-79
DUP    .SAV     41  13-Aug-79       RESORC.SAV     15  13-Aug-79
DIR    .SAV     17  13-Aug-79       RK     .SYS      3  13-Aug-79
EDIT   .SAV     19  13-Aug-79       DD     .SYS      5  13-Aug-79
SRCCOM.SAV      13  13-Aug-79       BINCOM.SAV     11  13-Aug-79
SLP    .SAV      9  13-Aug-79       SIPP   .SAV     14  13-Aug-79
 20 Files, 412 Blocks
 73 Free blocks
```

/DELETED This option lists a directory of files that have been deleted from a specific device, but whose file name information has not been destroyed. The listing includes the file names, types, sizes, creation dates, and starting block numbers in decimal of the files. The file names that print also represent tentative files. The listing can be useful in recovering files that have been accidentally deleted. Once you identify the file name and location, you can use the CREATE command or DUP to rename the area (see Section 8.2.1 for this procedure). The following command lists files on device DX0: that have been deleted.

```
.DIRECTORY/DELETED DX0:
 14-Dec-79
SYSGEN.CND     11  19-Nov-79  1403   TS     .MAC     2  27-Nov-79  2895
TM     .MAC    26  19-Nov-79  2926   MT     .SYS    32  27-Nov-79  3415
468DAT.DIR      1  14-Dec-79  3701   468DEL.DIR    527  14-Dec-79  3704
NUM2   .MAC     4  21-Nov-79  4231   NUM2   .LST   565  06-Sep-79  4235
 0 Files, 0 Blocks
 1164 Free blocks
```

Note in the example shown above that, since a deleted file does not really exist, the total number of files and blocks is 0.

/DOS Use this option to list the directory of a device that is in RSTS/E or DOS format. The only other options valid with /DOS are /BRIEF, /FAST, and /OWNER. The valid devices are DECtape for RSTS/E and DOS, and RK05 for DOS.

/END:n Use with /START:n and /BADBLOCKS to specify the last block number of a bad block scan. If you do not specify /END:n, the system scans to the last block on the volume.

/EXCLUDE This option lists a directory of all the files on a device except those files you specify. The following example lists all files on DX0: except the .SAV and .SYS files.

```
.DIRECTORY/EXCLUDE DX0:(*.SAV,*.SYS)
 29-Oct-79
RT11SJ.MAC     67P 06-Sep-79       RT11FB.MAC     80P 06-Sep-79
RT11BL.MAC     63P 06-Sep-79       DX     .MAC     3P 06-Sep-79
SWAP   .MAC    25P 06-Sep-79       TT     .MAC     2P 06-Sep-79
DP     .MAC     3P 06-Sep-79       DY     .MAC     4P 06-Sep-79
LP     .MAC     2P 06-Sep-79       RK     .MAC     3  06-Sep-79
STARTS.COM      1  27-Aug-79       DD     .MAC     5  06-Sep-79
 12 Files, 258 Blocks
 73 Free blocks
```

/FAST This option lists only file names and file types, omitting file lengths and associated dates. This is the same as /BRIEF.

/FILES Use this option with /BADBLOCKS to print the file names of bad blocks. If the system does not find any bad blocks, it prints only the heading, as the following example shows. Do not use this option if the volume is not a standard RT–11 directory-structured volume or if the volume does not contain an RT–11 directory.

```
.DIRECTORY/BADBLOCKS/FILES DT1:
?DUP-I-No bad blocks detected DT1:
```

/FREE Use this option to print a directory of unused areas and the size of each. This example lists the unused areas on device DK:.

```
.DIRECTORY/FREE
 14-Dec-79
< UNUSED >     11                  < UNUSED >      2
< UNUSED >     26                  < UNUSED >     32
< UNUSED >      1                  < UNUSED >    525
< UNUSED >      0                  < UNUSED >    565
 0 Files, 0 Blocks
 1162 Free blocks
```

/FULL This option lists the entire directory, including unused areas and their sizes in blocks (decimal). The following example lists the entire directory for device DX0:.

```
.DIRECTORY/FULL DX0:
 14-Dec-79
SWAP   .SYS     25P 23-Oct-79       RT11SJ.SYS     67P 23-Oct-79
RT11FB.SYS     80P 19-Nov-79       RT11BL.SYS     64P 19-Nov-79
TT     .SYS      2P 19-Nov-79       DT     .SYS      3P 19-Nov-79
DP     .SYS      3P 23-Oct-79       DX     .SYS      3P 19-Nov-79
DY     .SYS      4P 19-Nov-79       RF     .SYS      3P 19-Nov-79
RK     .SYS      3P 19-Nov-79       DL     .SYS      4P 23-Oct-79
DM     .SYS      5P 23-Oct-79       DS     .SYS      3P 19-Nov-79
DD     .SYS      5P 23-Oct-79       LP     .SYS      2P 23-Oct-79
LS     .SYS      2P 19-Nov-79       CR     .SYS      3P 19-Nov-79
MS     .SYS      9P 27-Nov-79       MTHD   .SYS      3P 23-Oct-79
DISMT1.COM      9P 27-Nov-79       MMHD   .SYS      4P 19-Nov-79
NUMBER.PAS      1  11-Dec-79       TONY   .AGP     14 17-Aug-79
NUM3   .LST     1  13-Dec-79       < UNUSED >    565
 25 Files, 322 Blocks
 164 Free blocks
```

/INTERCHANGE Use this option to list the directory of a diskette that is in interchange format. The only other options valid with /INTERCHANGE are /BRIEF and /FAST.

/NEWFILES This option includes in the directory listing only those files that were created on the current day. This is a convenient way to list the files you created in one session at the computer. The following command lists the new files on 19 May 1979.

```
.DIRECTORY/NEWFILES DT0:
 19-May-79
FILE1 .TXT      1  19-May-79        FILE2 .TXT    1  19-May-79
 2 Files, 2 Blocks
 328 Free blocks
```

/OCTAL This option lists the sizes (and starting block numbers if you also use /BLOCKS) in octal. If the device you specify is a magtape or cassette, the system prints the sequence numbers in octal. The following example shows an octal listing of device DX0:.

```
.DIRECTORY/OCTAL DX0:
 14-Dec-79 Octal
MYPROG.MAC      44P 12-Nov-79       TM     .MAC      31  27-Nov-79
VTMAC .MAC       7  18-Oct-79       SYSMAC.MAC      51  19-Nov-79
SWAP   .SYS     31  05-Sep-79       ANTON .MAC       4  19-Nov-79
RT11SJ.SYS     103  19-Nov-79       TT     .SYS      2  19-Nov-79
DX     .SYS      3  29-Aug-79       BUILD .MAC     144  19-Nov-79
 10 Files,    462 Blocks
    264 Free blocks
```

/ORDER[:category] This option sorts the directory of a device according to the category you specify. Table 4–3 summarizes the categories and their functions.

Table 4–3: Sort Categories

Category	Function
DATE	Sorts the directory chronologically by creation date. Files that have the same date are sorted alphabetically by file name and file type.
NAME	Sorts the directory alphabetically by file name. Files that have the same file name are sorted alphabetically by file type (this has the same effect as the /ALPHABETIZE option).
POSITION	Lists the files according to their position on the device (this is the same as using /ORDER with no category).
SIZE	Sorts the directory based on file size in blocks. Files that are the same size are sorted alphabetically by file name and file type.
TYPE	Sorts the directory alphabetically by file type. Files that have the same file type are sorted alphabetically by file name.

The following examples list the directory of device DX0:, according to each of the categories.

```
.DIRECTORY/ORDER:DATE DX0:
 14-Dec-79
BUILD .MAC    100  06-Sep-79       SYSMAC.MAC     41  19-Nov-79
DX    .SYS      3  06-Sep-79       TT     .SYS     2  19-Nov-79
MYPROG.MAC     36P 12-Oct-79       VTMAC  .MAC     7  19-Nov-79
RFUNCT.MAC      4  19-Nov-79       TM     .MAC    25  27-Nov-79
RT11SJ.SYS     67  19-Nov-79       SWAP   .SYS    25  05-Dec-79
 10 Files, 306 Blocks
 180 Free blocks


.DIRECTORY/ORDER:NAME DX0:
 14-Dec-79
BUILD .MAC    100  06-Sep-79       SWAP   .SYS    25  05-Dec-79
DX    .SYS      3  06-Sep-79       SYSMAC.MAC     41  19-Nov-79
MYPROG.MAC     36P 12-Oct-79       TM     .MAC    25  27-Nov-79
RFUNCT.SYS      4  19-Nov-79       TT     .SYS     2  19-Nov-79
RT11SJ.SYS     67  19-Nov-79       VTMAC  .MAC     7  19-Nov-79
 10 Files, 306 Blocks
 180 Free blocks


.DIRECTORY/ORDER:POSITION DX0:
 14-Dec-79
RT11SJ.SYS     67  19-Nov-79       BUILD .MAC    100  06-Sep-79
DX    .SYS      3  06-Sep-79       SYSMAC.MAC     41  19-Nov-79
MYPROG.MAC     36P 12-Oct-79       TM     .MAC    25  27-Nov-79
SWAP   .SYS    25  05-Dec-79       VTMAC  .MAC     7  19-Nov-79
RFUNCT.SYS      4  19-Nov-79       TT     .SYS     2  19-Nov-79
 10 Files, 306 Blocks
 180 Free blocks
```

```
.DIRECTORY/ORDER:SIZE DX0:
 14-Dec-79
TT     .SYS      2  19-Nov-79       SWAP   .SYS     25  05-Dec-79
DX     .SYS      3  06-Sep-79       MYPROG.MAC     36P 12-Oct-79
RFUNCT.SYS       4  19-Nov-79       SYSMAC.MAC     41  19-Nov-79
VTMAC .MAC       7  19-Nov-79       RT11SJ.SYS     67  19-Nov-79
TM     .MAC     25  27-Nov-79       BUILD  .MAC   100  06-Sep-79
 10 Files, 306 Blocks
 180 Free blocks


.DIRECTORY/ORDER:TYPE DX0:
 14-Dec-79
BUILD  .MAC    100  06-Sep-79       DX     .SYS      3  06-Sep-79
MYPROG.MAC     36P 12-Oct-79       RFUNCT.SYS       4  19-Nov-79
SYSMAC.MAC     41  19-Nov-79       RT11SJ.SYS     67  19-Nov-79
TM     .MAC     25  27-Nov-79       SWAP   .SYS     25  05-Dec-79
VTMAC .MAC       7  19-Nov-79       TT     .SYS      2  19-Nov-79
 10 Files, 306 Blocks
 180 Free blocks
```

/OUTPUT:filespec Use this option to specify a device and file name for the output listing file. Normally, the directory listing appears on the console terminal. If you omit the file type for the listing file, the system uses .DIR.

/OWNER:[nnn,nnn] Use this option with /DOS to specify a user identification code (UIC). Note that the square brackets are part of the UIC; you must type them.

/POSITION Use this option to list the file sequence numbers of files stored on a magtape. See /COLUMNS:n for a sample listing.

/PRINTER Use this option to print the directory listing on the line printer. The default output device is the terminal. Note that the /PRINTER option does not use the QUEUE program to queue the directory listing.

/REVERSE This option lists a directory in the reverse order of the sort you specify with /ALPHABETIZE, /ORDER, or /SORT. The following example sorts the directory of DX0: and lists it in reverse order by size.

```
.DIRECTORY/ORDER:SIZE/REVERSE DX0:
 14-Dec-79
BUILD  .MAC    100  06-Sep-79       TM     .MAC     25  27-Nov-79
RT11SJ.SYS     67  19-Nov-79       VTMAC .MAC       7  19-Nov-79
SYSMAC.MAC     41  19-Nov-79       RFUNCT.SYS       4  19-Nov-79
MYPROG.MAC     36P 12-Oct-79       DX     .SYS      3  06-Sep-79
SWAP   .SYS    25  05-Dec-79       TT     .SYS      2  19-Nov-79
 10 Files, 306 Blocks
 180 Free blocks
```

/SINCE[date] This option lists a directory of all files on a specified device that were created on or after a specified date. The following command lists only those files on DK: that were created on or after 13 August 1977.

```
*DIRECTORY/SINCE:13:AUG:79
 14-Dec-79
RT11SJ.SYS     67P 14-Aug-79       RT11FB.SYS     80P 02-Sep-79
RT11BL.SYS     63P 19-Aug-79       DX     .SYS     3P 10-Sep-79
SWAP  .SYS     25P 02-Sep-79       TT     .SYS     2P 15-Sep-79
SIPP  .SAV     14  02-Sep-79
 7 Files, 154 Blocks
 332 Free blocks
```

/SORT[:category] This option sorts the directory of a device according to the category you specify. It is the same as /ORDER[:category].

/START[:n] Use this option with the /BADBLOCKS option to specify the starting block, and optionally the last block if you use /END:n, of the bad block scan. The argument *n* represents a block number in decimal. If you do not supply a value with /START, the system scans from the first block on the volume. If you do not specify /END:n, the system scans to the end of the volume.

/SUMMARY This option lists a summary of the device directory. The summary lists the number of files in each segment and the number of segments in use on the volume you specify. The /SUMMARY option does not list the segments in numerical order, only the order in which they are linked on the volume. The following example lists the summary of the directory for device DK:.

```
*DIRECTORY/SUMMARY
 14-Nov-79

    44 Files in segment 1

    46 Files in segment 4

    37 Files in segment 2

    34 Files in segment 5

    38 Files in segment 3

    16 Available segments, 5 in use

199 Files, 3647 Blocks
1115 Free blocks
```

/TERMINAL This option lists directory information on the console terminal. This is the default operation.

/TOPS Use this option to list the directory of a DECtape that is in DECsystem–10 format. The only other options valid with /TOPS are /BRIEF and /FAST.

/VERIFY Use this option with the /BADBLOCKS option to read a bad block, write to it, and read it again. If the system can not read the block, it reports a hard error. If the block recovers, it reports a soft error. This procedure does not destroy data already on the volume.

Use this option only when necessary; DIGITAL does not guarantee the integrity of the data recovered from a soft bad block error.

/VOLUMEID[:ONLY] Use /VOLUMEID to print the volume ID and owner name along with the directory listing of the storage volume. If you include the optional argument, ONLY, the system prints only the volume ID and owner name.

The following example displays the volume ID of volume DK1:

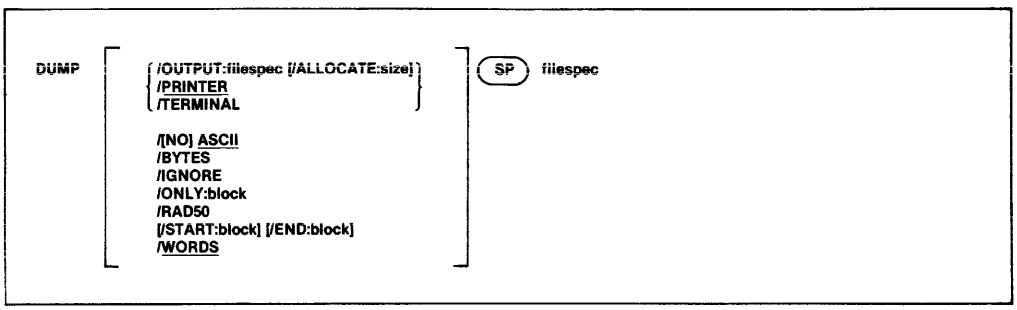
```
.DIRECTORY/VOLUMEID DX1:
 14-Dec-79
 Volume ID: BACKUP2
 Owner    : Marcy
SWAP  .SYS    25P 19-Nov-79      RT11SJ.SYS    67P 19-Nov-79
RT11FB.SYS    80P 19-Nov-79      RT11BL.SYS    64P 19-Nov-79
TT    .SYS     2P 19-Nov-79      DT    .SYS     3P 19-Nov-79
DP    .SYS     3P 19-Nov-79      DX    .SYS     3P 19-Nov-79
DY    .SYS     4P 19-Nov-79      RF    .SYS     3P 19-Nov-79
RK    .SYS     3P 19-Nov-79      DL    .SYS     4P 19-Nov-79
 12 Files, 271 Blocks
 215 Free blocks
```

/WAIT Use with the /BADBLOCKS option when you want the system to initiate a bad block scan but to pause for you to mount the input volume. This option is particularly useful if you have a single-disk system. When you use this option, and the system volume is mounted, the system initiates the operation you specify, then prints *Mount input volume in <device>; Continue?*. The prompt *<device>* represents the device into which you mount the volume. Mount your input volume and type Y, followed by a carriage return.

The following sample performs a bad block scan on an RK05 disk.

```
DIRECTORY/WAIT/BADBLOCKS RK0:
Mount input volume in RK0: Continue? Y
?DUP-I-No bad blocks detected RK0:
Mount system volume in RK0: Continue? Y
```

The DUMP command can print on the terminal or line printer, or write to a file all or any part of a file in octal words, octal bytes, ASCII characters, and/or Radix–50 characters. It is particularly useful for examining directories and files that contain binary data.



In the command syntax shown above, *filespec* represents the device or file you want to examine. If you do not specify an output file, the listing prints on the line printer. If you do not specify a file type for an output file, the system uses .DMP. You can specify the entire command on one line, or you can rely on the system to prompt you for information. The DUMP command prompt is *Device or file?*.

Notice that some of the options (/ONLY, /START, and /END) accept a block number as an argument. Remember that all block numbers are in octal, and that the first block of a device or file is block 0. To specify a decimal block number, follow the number with a decimal point. If you are dumping a file, the block numbers you specify are relative to the beginning of that file. If you are dumping a device, the block numbers are the absolute (physical) block numbers on that device.

The system handles operations involving magtape and cassette differently from operations involving random-access devices. If you dump an RT–11 file-structured tape and specify only a device name in the file specification, the system reads only as far as the logical end-of-tape. Logical end-of-tape is indicated by an end-of-file label (EOF1) followed by two tape marks. For non-file-structured tape, logical end-of-tape is indicated by two consecutive tape marks. If you dump a cassette and specify only the device name in the file specification, the results are unpredictable. For magtape dumps, tape mark messages appear in the output listing as the system encounters them on the tape.

NOTE

The DUMP operation does not print data from track 0 of diskettes.

The following sections describe the options you can use with the DUMP command. Following the options are some sample listings and an explanation of how to interpret them.

/ALLOCATE:size Use this option with /OUTPUT to reserve space on the device for the output listing file. The argument *size* represents the number of blocks of space to allocate. The meaningful range for this value is from 1 to 32767. A value of -1 is a special case that creates the largest file possible on the device.

/ASCII This option prints the ASCII equivalent of each octal word or byte that is dumped. A dot (.) represents characters that are not printable. This is the default operation.

/NOASCII Use this option to suppress the ASCII output, which appears in the right hand column of the listing (or below the bytes if you have specified /BYTES). This allows the listing to fit in 72 columns.

/BYTES Use this option to display information in octal bytes. The system does not display words unless you also use /WORDS.

/END:block Use this option to specify an ending block number for the dump. The system dumps the device or file you specify, beginning with block 0 (unless you use /START) and continuing until it dumps the block you specify with /END.

/FOREIGN Use this option to dump a magtape that is not RT–11 file-structured.

/IGNORE Use this option to ignore errors that occur during a dump operation. Use /IGNORE if an input or output error occurred when you tried to perform a normal dump operation.

/ONLY:block Use this option to dump only the block number you specify.

/OUTPUT:filespec Use this option to specify a device and file name for the output listing file. Normally, the listing appears on the line printer. If you omit the file type for the listing file, the system uses .DMP.

/PRINTER This option causes the output listing to appear on the line printer. This is the default operation.

/RAD50 This option prints the Radix–50 equivalent of each octal word that is dumped.

/START:block Use this option to specify a starting block number for the dump. The system dumps the device or file, beginning at the block number you specify with /START and continuing to the end of the device or file (unless you use /END).

/TERMINAL This option causes the output listing to appear on the console terminal. Normally, the listing appears on the line printer.

/WORDS This option displays information in octal words. This is the default operation.

The following command dumps block 1 of the file SYSMAC.MAC. The output listing, which shows octal bytes and their ASCII equivalent, is stored in file MACLIB.DMP. The PRINT command prints the contents of the file on the line printer.

```
.DUMP/OUTPUT:MACLIB/BYTES/ONLY:1 SYSMAC.MAC

.PRINT MACLIB.DMP


SY:SYSMAC.MAC
BLOCK NUMBER  00001
000/ 120 040 117 106 040 040 124 110 105 040 040 123 117 106 124 127
     P       O   F           T   H   E           S   O   F   T   W
020/ 101 122 105 040 040 111 123 040 040 110 105 122 105 102 131 015
     A   R   E           I   S           H   E   R   E   B   Y   .
040/ 012 073 040 124 122 101 116 123 106 105 122 122 105 104 056 015
     .   ;       T   R   A   N   S   F   E   R   R   E   D   .   .
060/ 012 073 015 012 073 040 124 110 105 040 111 116 106 117 122 115
     .   ;   .   .   ;       T   H   E       I   N   F   O   R   M
100/ 101 124 111 117 116 040 111 116 040 124 110 111 123 040 123 117
     A   T   I   O   N       I   N       T   H   I   S       S   O
120/ 106 124 127 101 122 105 040 111 123 040 123 125 102 112 105 103
     F   T   W   A   R   E       I   S       S   U   B   J   E   C
140/ 124 040 124 117 040 103 110 101 116 107 105 040 040 127 111 124
     T       T   O       C   H   A   N   G   E           W   I   T
160/ 110 117 125 124 040 040 116 117 124 111 103 105 015 012 073 040
     H   O   U   T           N   O   T   I   C   E   .   .   ;
200/ 101 116 104 040 040 123 110 117 125 114 104 040 040 116 117 124
     A   N   D           S   H   O   U   L   D           N   O   T
220/ 040 040 102 105 040 040 103 117 116 123 124 122 125 105 104 040
         B   E           C   O   N   S   T   R   U   E   D
240/ 040 101 123 040 040 101 040 103 117 115 115 111 124 115 105 116
         A   S           A       C   O   M   M   I   T   M   E   N
260/ 124 040 102 131 040 104 111 107 111 124 101 114 040 105 121 125
     T       B   Y       D   I   G   I   T   A   L       E   Q   U
300/ 111 120 115 105 116 124 015 012 073 040 103 117 122 120 117 122
     I   P   M   E   N   T   .   .   ;       C   O   R   P   O   R
320/ 101 124 111 117 116 056 015 012 073 015 012 073 040 104 111 107
     A   T   I   O   N   .   .   .   ;   .   .   ;       D   I   G
340/ 111 124 101 114 040 101 123 123 125 115 105 123 040 116 117 040
     I   T   A   L       A   S   S   U   M   E   S       N   O
360/ 122 105 123 120 117 116 123 111 102 111 114 111 124 131 040 106
     R   E   S   P   O   N   S   I   B   I   L   I   T   Y       F
400/ 117 122 040 124 110 105 040 125 123 105 040 117 122 040 040 122
     O   R       T   H   E       U   S   E       O   R           R
420/ 105 114 111 101 102 111 114 111 124 131 040 040 117 106 040 040
     E   L   I   A   B   I   L   I   T   Y           O   F
440/ 111 124 123 015 012 073 040 123 117 106 124 127 101 122 105 040
     I   T   S   .   .   ;       S   O   F   T   W   A   R   E
460/ 117 116 040 105 121 125 111 120 115 105 116 124 040 127 110 111
     O   N       E   Q   U   I   P   M   E   N   T       W   H   I
500/ 103 110 040 111 123 040 116 117 124 040 123 125 120 120 114 111
     C   H       I   S       N   O   T       S   U   P   P   L   I
520/ 105 104 040 102 131 040 104 111 107 111 124 101 114 056 015 012
     E   D       B   Y       D   I   G   I   T   A   L   .   .   .
540/ 014 056 115 101 103 122 117 040 056 056 126 061 056 056 015 012
     .   .   M   A   C   R   O       .   .   V   1   .   .   .   .
560/ 056 115 103 101 114 114 011 056 056 056 103 115 060 054 056 056
     .   M   C   A   L   L   .   .   .   .   C   M   0   ,   .   .
600/ 056 103 115 061 054 056 056 056 103 115 062 054 056 056 056 103
     .   C   M   1   ,   .   .   .   C   M   2   ,   .   .   .   C
620/ 115 063 054 056 056 056 103 115 064 054 056 056 056 103 115 065
     M   3   ,   .   .   .   C   M   4   ,   .   .   .   C   M   5
640/ 054 056 056 056 103 115 066 015 012 056 056 056 126 061 075 061
     ,   .   .   .   C   M   6   .   .   .   .   .   V   1   =   1
660/ 015 012 056 105 116 104 115 015 012 015 012 056 115 101 103 122
     .   .   .   E   N   D   M   .   .   .   .   .   M   A   C   R
700/ 117 040 056 056 126 062 056 056 015 012 056 115 103 101 114 114
     O       .   .   V   2   .   .   .   .   .   M   C   A   L   L
720/ 011 056 056 056 103 115 060 054 056 056 056 103 115 061 054 056
     .   .   .   .   C   M   0   ,   .   .   .   C   M   1   ,   .
740/ 056 056 103 115 062 054 056 056 056 103 115 063 054 056 056 056
     .   .   C   M   2   ,   .   .   .   C   M   3   ,   .   .   .
760/ 103 115 064 054 056 056 056 103 115 065 054 056 056 056 103 115
     C   M   4   ,   .   .   .   C   M   5   ,   .   .   .   C   M
```

In the printout above, the heading shows which file was dumped and which block of the file follows. The numbers in the leftmost column indicate the byte offset from the beginning of the block. Remember that these are all octal values, and that there are two bytes per word. The octal bytes that were dumped appear in the next eight columns. The ASCII equivalent of each octal byte appears underneath the byte. The system substitutes a dot (.) for nonprinting codes, such as those for control characters.

The last example shows block 6 (the directory) of device RK0:. The output is in octal words with Radix–50 equivalents below each word.

```
.DUMP/NOASCII/RAD50/ONLY:6 RK0:

RK0:/N/X/O:6
BLOCK NUMBER  00006
000/ 000020 000002 000004 000000 000046 002000 075131 062000
       P      B      D                8     YX     SWA    P
020/ 075273 000031 000000 027147 002000 071677 142302 075273
     SYS      Y             GP9     YX     RT1    1SJ    SYS
040/ 000103 000000 027147 002000 071677 141262 075273 000120
      AS            GP9     YX     RT1    1FB    SYS     B
060/ 000000 027147 002000 071677 141034 075273 000100 000000
            GP9     YX     RT1    1BL    SYS     AX
100/ 027147 002000 100040 000000 075273 000002 000000 027147
     GP9     YX     TT              SYS      B             GP9
120/ 002000 016040 000000 075273 000003 000000 027147 002000
      YX     DT             SYS      C             GP9     YX
140/ 015600 000000 075273 000003 000000 027147 002000 016300
     DP             SYS      C             GP9     YX     DX
160/ 000000 075273 000003 000000 027147 002000 016350 000000
            SYS      C             GP9     YX     DY
200/ 075273 000004 000000 027147 002000 070560 000000 075273
     SYS      D             GP9     YX     RF             SYS
220/ 000003 000000 027147 002000 071070 000000 075273 000003
       C            GP9     YX     RK             SYS      C
240/ 000000 027147 002000 015340 000000 075273 000004 000000
            GP9     YX     DL              SYS      D
260/ 027147 002000 015410 000000 075273 000005 000000 027147
     GP9     YX     DM              SYS      E             GP9
300/ 002000 015770 000000 075273 000003 000000 027147 002000
      YX     DS             SYS      C             GP9     YX
320/ 014640 000000 075273 000005 000000 027147 002000 046600
     DD             SYS      E             GP9     YX     LP
340/ 000000 075273 000002 000000 027147 002000 046770 000000
            SYS      B             GP9     YX     LS
360/ 075273 000002 000000 027147 002000 012620 000000 075273
     SYS      B             GP9     YX     CR             SYS
400/ 000003 000000 027147 002000 052070 000000 075273 000011
       C            GP9     YX     MS             SYS      I
420/ 000000 027547 002000 052150 014400 075273 000003 000000
            GWO     YX     MTH     D      SYS      C
440/ 027147 002000 015173 052177 012445 000011 000000 027547
     GP9     YX     DIS    MT1    COM      I             GWO
460/ 002000 051520 014400 075273 000004 000000 027147 002000
      YX     MMH    D      SYS      D             GP9     YX
500/ 015173 052200 012445 000010 000000 027547 002000 052100
     DIS    MT2    COM      H             GWO     YX     MSH
520/ 014400 075273 000004 000000 027147 002000 054540 000000
     D      SYS      D             GP9     YX     NL
540/ 075273 000002 000000 027147 002000 062170 000000 075273
     SYS      B             GP9     YX     PC             SYS
560/ 000002 000000 027147 002000 062240 000000 075273 000003
       B            GP9     YX     PD             SYS      C
600/ 000000 027147 002000 012740 000000 075273 000005 000000
            GP9     YX     CT              SYS      E
620/ 027147 002000 006250 000000 075273 000007 000000 027147
     GP9     YX     BA              SYS      G             GP9
```

```
640/ 002000 016130 000000 073376 000051 000000 027147 002000
      YX     DUP           SAV      AA            GP9      YX
660/ 023752 050574 073376 000023 000000 027147 002000 070533
     FOR     MAT     SAV      S           GP9      YX     RES
700/ 060223 073376 000017 000000 027147 002000 015172 000000
     ORC     SAV      O           GP9      YX     DIR
720/ 073376 000021 000000 027147 002000 075273 050553 074324
     SAV      Q           GP9      YX     SYS     MAC     SML
740/ 000052 000000 027147 002000 017751 076400 073376 000023
      AB           GP9      YX     EDI     T       SAV      S
760/ 000000 027147 002000 042614 000000 073376 000073 000000
            GP9      YX     KED           SAV      AS
```

E

The E (Examine) command prints in octal the contents of an address on the console terminal.

E (SP) address [-address]

In the command syntax illustrated above, *address* represents an octal address that, when added to the relocation base value from the Base command, provides the actual address that the system examines. This command permits you to open specific locations in memory and inspect their contents. It is most frequently used after a GET command to examine locations in a program.

The Examine command accepts both word and byte addresses, but it always executes the command as though you specified a word address. (If you specify an odd address, the system decreases it by one.)

If you specify more than one address (in the form *address1-address2)*, the sys- tem prints the contents of *address1* through *address2*, inclusive. The second address (*address2*) must always be greater than the first address. If you do not specify an address, the system prints the contents of relative location 0.

Note that you cannot examine addresses outside the background.

The following example prints the contents of location 1000, assuming the relocation base is 0.

```
.E 1000
127401
```

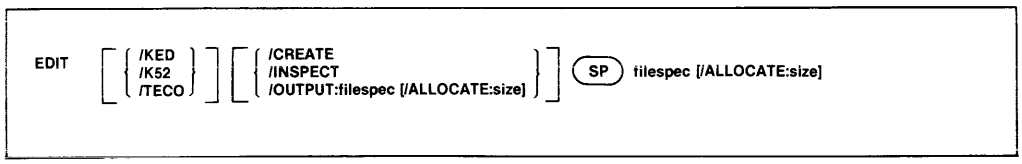
The next command sets the relocation base to 1000.

```
.B 1000
```

The following command prints the contents of locations 2000 (offset of 1000 from last B command) through 2005.

```
.E 1001-1005
127401 007624 127400
```

The EDIT command invokes the text editor.



The text editor, EDIT, is a program that creates or modifies ASCII files for use as input to programs such as the MACRO assembler or the FORTRAN compiler. The editor reads ASCII files from any input device, makes specified changes, and writes the file on an output device. It also allows efficient use of VT11 or VS60 display hardware, if this is part of the system configuration (except in multi-terminal systems).

You can also use the Keypad Editor (KED for VT100 terminals, K52 for VT52 terminals) as an alternative to EDIT. The Keypad Editor is restricted to the VT100 and VT52 terminals, however. You can invoke the Keypad Editor with the /KED or /K52 options described below. For more information on the Keypad Editor, see the *PDP–11 Keypad Editor User's Guide.*

NOTE

You can use the SET EDIT command to set a default editor (EDIT, KED, or K52) so that when you issue the EDIT command, you invoke that editor. The system defaults to the EDIT editor each time you bootstrap, however. For more details, see the SET EDIT command description.

EDIT considers a file to be divided into logical units called pages. A page of text is generally 50–60 lines long (delimited by form feed characters) and corresponds approximately to a physical page of a program listing. EDIT reads one page of text at a time from the input file into its internal buffers where the page becomes available for editing. You can then use editing commands to:

- Locate text to be changed
- Execute and verify the changes
- List an edited page on the console terminal
- Output a page of text to the output file

In the command syntax illustrated above, *filespec* represents the file you wish to edit. You can enter the EDIT command on one line, or you can rely on the system to prompt you for information. If you do not supply a file specification for the file to edit, the system prompts *File?*. If you do not specify any option with the EDIT command, the text editor performs the edit backup operation. To do this, it changes the name of the original file, giving it a file

type of .BAK when you finish making your editing changes. The actual file renaming occurs when you successfully exit from the editor.

When you want to edit an existing file, the editor does not perform any I/O operation as a result of your command. You must issue the R command to the editor to read the first page of text and make it available for you to work on. The following example invokes EDIT, opens an existing file, and reads the first page of text:

```
.EDIT MYFILE.TXT
*R$$
```

When you issue an EDIT command, the system invokes the text editor. (You can use the SET EDIT command to set the default editor. If you do not use the SET EDIT command, the system assumes EDIT.SAV each time you issue the EDIT command. See the SET EDIT command for more information.) It is possible to receive an error or warning message as a result of the EDIT command. If, for example, the file you need to edit with EDIT does not exist on device DK:, the editor issues an error message and remains in control. For example:

```
.EDIT/INSPECT EXAMP3.TXT
?EDIT-F-File not found
*^C$$
```

When a situation like this occurs, you can either issue another command directly to the text editor or enter CTRL/C followed by two ESCAPEs to return control to the monitor.

NOTE

To perform any edit operations on a protected file, you must disable the file's protected status (see the RENAME command description).

The following sections describe the options you can use with the EDIT command. A complete description of EDIT is contained in Chapter 5.

/ALLOCATE:size Use this option with /OUTPUT or after the file specification to reserve space on the device for the output file. The argument *size* represents the number of blocks of space to allocate. The meaningful range for this value is from 1 to 32767. A value of -1 is a special case that creates the largest file possible on the device.

/CREATE Use this option to build a new file. With EDIT you can also create a new file while you are working with the text editor by using the EDIT Write (EW) command, described in Chapter 5. The following example creates a file called NEWFIL.TXT on device DK:, inserts one line of text, and then closes the file.

```
.EDIT/CREATE NEWFIL.TXT
*ITHIS IS A NEW FILE.
$$
*EX$$
```

To create a file with /KED or /K52, see the *PDP–11 Keypad Editor User's Guide.*

/EXECUTE:filespec Use this option with /TECO to execute the TECO commands contained in the file you specify with /EXECUTE.

/INSPECT Use this option to open a file for reading. This option does not create any new output files. You can also open a file for inspection while you are working with the EDIT by using the Edit Read (ER) command, which is explained in Chapter 5.

The following command opens an existing file for inspection, lists its contents, and then exits.

```
.EDIT/INSPECT NEWFIL.TXT
*R$$
*/L$$
THIS IS A NEW FILE.
*^C$$
```

/KED This option invokes the Keypad Editor (KED). For more information on the Keypad Editor, see the *PDP–11 Keypad Editor User's Guide.* Use /KED only if you are using a VT100 terminal.

/K52 This option invokes the Keypad Editor. Use /K52 only if you are using a VT52 terminal. For more information on the Keypad Editor, see the *PDP–11 Keypad Editor User's Guide.*

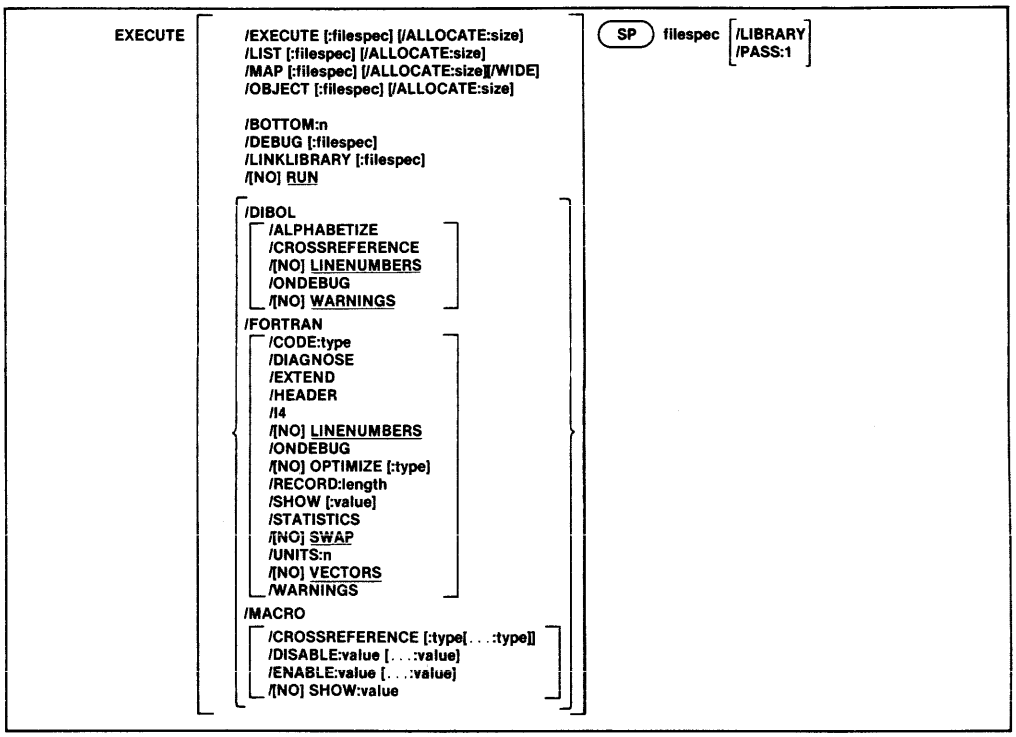
/OUTPUT:filespec This option directs the text you edit to the file you specify, leaving the input file unchanged. You can also write text to an output file while you are working with EDIT by using the Edit Write (EW) command, explained in Chapter 5. The following command reads file ORIG.TXT, and writes the edited text to file CHANGE.TXT.

```
.EDIT/OUTPUT:CHANGE.TXT ORIG.TXT
*
```

/TECO This option invokes the TECO editor. (TECO is not supported by DIGITAL. It is distributed on the RT–11 kit for the convenience of those customers who normally order TECO from the DECUS Program Library) For more information on TECO see the *PDP–11 TECO User's Guide.*

EXECUTE

The EXECUTE command invokes one or more language processors to assemble or compile the files you specify. It also links object modules and initiates execution of the resultant program.



In the command line shown above, *filespecs* represents one or more files to be included in the assembly. The default file types for the output files are .LST for listing files, .MAP for load map files, .OBJ for object files, and .SAV for memory image files. The defaults for input files depend on the language processor involved. These defaults include .MAC for MACRO files, .FOR for FORTRAN files, and .DBL for DIBOL files.

To compile (or assemble) multiple source files into a single object file, separate the files by plus (+) signs in the command line. Unless you specify otherwise, the system creates an object file with the same name as the first input file and gives it an .OBJ file type.

To compile multiple files in independent compilations, separate the files by commas (,) in the command line. This generates a corresponding object file for each set of input files. The system then links together all the object files and creates a single executable file. You can combine up to six files for a compilation producing a single object file. You can specify the entire EXECUTE command as one line, or you can rely on the system to prompt you for information. The EXECUTE command prompt is *Files?*.

There are several ways to establish which language processor the EXECUTE command invokes:

1. Specify a language-name option, such as /MACRO, to invoke the MACRO assembler.

2. Omit the language-name option and explicitly specify the file type for the source files. The EXECUTE command then invokes the language processor that corresponds to that file type. Specifying the file SOURCE.MAC, for example, invokes the MACRO assembler.

3. Let the system choose a file type of .MAC, .DBL, or .FOR for the source file you name. The handler for the device you specify must be loaded. If you specify DX1:A, and the DX handler is loaded, the system searches for source files A.MAC and A.DBL, in that order. If it finds one of these files, the system invokes the corresponding language processor. If it cannot find one of these files, or if the device handler associated with the input file is not resident, the system assumes a file type of .FOR and invokes the FORTRAN compiler.

   If the language processor selected as a result of this procedure described above is not on the system device (SY:), the system issues an error message.

Language options are position-dependent. That is, they have different meanings depending on where you place them in the command line. Options that qualify a command name apply across the entire command string. Options that follow a file specification apply only to the file (or group of files separated by plus signs) that they follow in the command string.

The following sections describe the options you can use with the EXECUTE command.

/ALLOCATE:size Use this option with /EXECUTE, /LIST, /MAP, or /OBJECT to reserve space on the device for the output file. The argument *size* represents the number of blocks of space to allocate. The meaningful range for this value is from 1 to 32767. A value of -1 is a special case that creates the largest file possible on the device.

/ALPHABETIZE Use this option with /DIBOL to alphabetize the entries in the symbol table listing. This is useful for program maintenance and debugging.

/BOTTOM:n Use this option to specify the lowest address to be used by the relocatable code in the load module. The argument *n* represents a six-digit, unsigned, even octal number. If you do not use this option, the system positions the load module so that the lowest address is location 1000 (octal). This option is invalid for foreground links.

/CODE:type Use this option with /FORTRAN to produce object code that is designed for a particular hardware configuration. The argument *type* represents a three-letter abbreviation for the type of code to be produced. The valid values are: EAE, EIS, FIS, and THR. See the *RT–11/RSTS/E FOR-*

TRAN IV User's Guide for a complete description of the types of code and their function.

/CROSSREFERENCE[:type[...:type]] Use this option with /MACRO or /DIBOL to generate a symbol cross-reference section in the listing. This information is useful for program maintenance and debugging. Note that the system does not generate a listing by default. You must also specify /LIST in the command line to get a cross-reference listing.

With MACRO this option takes an optional argument. The argument *type* represents a one-character code that indicates which sections of the cross-reference listing the assembler should include. Table 4–10 summarizes the valid arguments and their meaning.

/DEBUG[:filespec] Use this option to link ODT (on-line debugging technique, described in Chapter 21) with your program to help you debug it. If you supply the name of another debugging program, the system links the debugger you specify with your program. The debugger is always linked low in memory relative to your program.

/DIAGNOSE Use this option with /FORTRAN to help analyze an internal compiler error. /DIAGNOSE expands the crash dump information to include internal compiler tables and buffers. Submit the diagnostic printout to DIGITAL with a software performance report (SPR) form. The information in the listing can help DIGITAL programmers locate the compiler error and correct it.

/DIBOL This option invokes the DIBOL language processor to compile the associated files.

/DISABLE:value[...:value] Use this option with /MACRO to specify a .DSABL directive. Table 4–11 summarizes the arguments and their meaning. See the *PDP–11 MACRO Language Reference Manual* for a description of the directive and a list of all legal values.

/ENABLE:value[...:value] Use this option with /MACRO to specify an .ENABL directive. Table 4–11 summarizes the arguments and their meaning. See the *PDP–11 MACRO Language Reference Manual* for a description of the directive and a list of all legal values.

/EXECUTE[:filespec] Use this option to specify a file name or device for the executable file. Because the EXECUTE command creates executable files by default, the following two commands have the same meaning:

```
*EXECUTE  MYPROG

*EXECUTE/EXECUTE  MYPROG
```

Both commands link MYPROG.OBJ and produce MYPROG.SAV as a result. The /EXECUTE option has different meanings when it follows the command and when it follows the file specification. The following command creates an executable file called PROG1.SAV on device RK1:.

```
*EXECUTE/EXECUTE:RK1:  PROG1,PROG2
```

The next command creates an executable file called MYPROG.SAV on device DK:.

```
.EXECUTE RTN1,RTN2,MYPROG/EXECUTE
```

/EXTEND Use this option with /FORTRAN to change the right margin for source input lines from column 72 to column 80.

/FORTRAN This option invokes the FORTRAN language processor to compile the associated files.

/HEADER Use this option with /FORTRAN to include in the printout a list of options currently in effect.

/I4 Use this option with /FORTRAN to allocate two words for the default integer data type (FORTRAN uses only one-word integers) so that it takes the same physical space as real variables.

/LIBRARY Use this option with /MACRO to identify the file the option qualifies as a macro library file. Use it only after a library file specification in the command line. The MACRO assembler looks first to the library associated with the most recent /LIBRARY option to satisfy references (made with the .MCALL directive) from MACRO programs. It then looks to any libraries you specified earlier in the command line, and it looks last to SYSMAC.SML.

In the example below, the two files A.FOR and B.FOR are compiled together, producing B.OBJ and B.LST. The MACRO assembler assembles C.MAC, satisfying .MCALL references from MYLIB.MAC and SYSMAC.SML. It produces C.OBJ and C.LST. The system then links B.OBJ and C.OBJ together, resolving undefined references from SYSLIB.OBJ, and produces the executable file B.SAV. Finally, the system loads and executes B.SAV.

```
.EXECUTE A+B/LIST/OBJECT,MYLIB/LIBRARY+C.MAC/LIST/OBJECT
```

/LINENUMBERS Use this option with /DIBOL or /FORTRAN to include internal sequence numbers in the executable program. These are especially useful in debugging programs. This is the default operation.

/NOLINENUMBERS Use this option with /DIBOL or /FORTRAN to suppress the generation of internal sequence numbers in the executable program. This produces a smaller program and optimizes execution speed. Use this option to compile only those programs that are already debugged; otherwise the line numbers in DIBOL or FORTRAN error messages are difficult to interpret.

/LINKLIBRARY:filespec Use this option to include the library file name you specify as an object module library during the linking operation. Repeat the option if you need to specify more than one library file.

/LIST[:filespec] You must specify this option to produce a compilation or assembly listing. The /LIST option has different meanings depending on where you put it in the command line.

If you specify /LIST without *filespec* in the list of options that immediately follows the command name, the system generates a listing that prints on the line printer. If you follow /LIST with a device name, the system creates a listing file on that device. If the device is a file-structured device, the system stores the listing file on that device, assigning it the same name as the input file with a .LST file type. The following command produces a listing on the terminal:

```
.EXECUTE/LIST:TT A.FOR
```

The next command creates a listing file called A.LST on RK3:.

```
.EXECUTE/LIST:RK3: A.MAC
```

If the /LIST option contains a name and file type to override the default of .LST, the system generates a listing file with that name. The following command, for example, compiles A.FOR and B.FOR together, producing files A.OBJ and FILE1.OUT on device DK:. It then links A.OBJ (using SYSLIB.OBJ as needed) and produces A.SAV.

```
.EXECUTE/NORUN/FORTRAN/LIST:FILE1.OUT A+B
```

Another way to specify /LIST is to type it after the file specification to which it applies. To produce a listing file with the same name as a particular input file, you can use a command similar to this one:

```
.EXECUTE/DIBOL A+B/LIST:RK3:
```

The command shown above compiles A.DBL and B.DBL together, producing files DK:A.OBJ and RK3:B.LST. It then links A.OBJ (using SYSLIB.OBJ as needed) and produces DK:A.SAV. If you specify a file name on a /LIST option following a file specification in the command line, it has the same meaning as when it follows the command. The following two commands have the same results:

```
.EXECUTE/MACRO A/LIST:B

.EXECUTE/MACRO/LIST:B A
```

Remember that file options apply only to the file (or group of files that are separated by plus signs) that they follow in the command string. For example:

```
.EXECUTE/NORUN A.MAC/LIST+B.FOR
```

This command compiles A.MAC, producing A.OBJ and A.LST. It also compiles B.FOR, producing B.OBJ. However, it does not produce any listing file for the compilation of B.FOR. Finally, the system links A.OBJ and B.OBJ together, producing A.SAV.

/MACRO This option invokes the MACRO assembler to assemble associated files.

/MAP[:filespec] You must specify this option to produce a load map after a link operation. The /MAP option has different meanings depending on

where you put it in the command line. It follows the same general rules outlined above for /LIST.

/OBJECT[;filespec] Use this option to specify a file name or device for the object file. Because the EXECUTE command creates object files by default, the following two commands have the same meaning:

```
•EXECUTE/FORTRAN A

•EXECUTE/FORTRAN/OBJECT A
```

Both commands compile A.FOR and produce A.OBJ as output. The /OBJECT option functions like the /LIST option; it can be either a command or a file qualifier.

As a command option, /OBJECT applies across the entire command string. The following command, for example, assembles A.MAC and B.MAC separately, creating object files A.OBJ and B.OBJ on RK1:.

```
•EXECUTE/OBJECT:RK1: A.MAC,B.MAC
```

Use /OBJECT as a file option to create an object file with a specific name or destination. The following command compiles A.DBL and B.DBL together, creating files B.LST, B.OBJ, and B.SAV.

```
•EXECUTE/DIBOL A+B/LIST/OBJECT/EXECUTE
```

/ONDEBUG Use this option with /DIBOL to include a symbol table in the object file. You can then use a debugging program to find and correct errors in the object file.

Use /ONDEBUG with /FORTRAN to include debug lines (those that have a D in column one) in the compilation. You do not, therefore, have to edit the file to include these lines in the compilation or to logically remove them. You can include messages, flags, and conditional branches to help you trace program execution and find an error.

/OPTIMIZE:type Use this option with /FORTRAN to enable certain options that optimize object code for various conditions. The argument *type* represents the three-letter code for the type of optimization to be enabled. Table 4–4 summarizes the codes and their meaning. This option is not available with version 2.5 of the FORTRAN compiler.

/NOOPTIMIZE:type Use this option with /FORTRAN to disable certain options that optimize object code for various conditions. The argument *type* represents the three-letter code for the type of optimization to be disabled. Table 4–4 summarizes the codes and their meaning. This option is not available with version 2.5 of the FORTRAN compiler.

/PASS:1 Use this option with /MACRO on a prefix macro file to process that file only during pass 1 of the assembly. This option is useful when you assemble a source program together with a prefix file that contains only macro definitions, since these do not need to be redefined in pass 2 of the

assembly. The following command assembles a prefix file and a source file together, producing files PROG1.OBJ, PROG1.LST, and PROG1.SAV.

```
.EXECUTE/NORUN/MACRO PREFIX/PASS:1+PROG1/LIST/OBJECT/EXECUTE
```

/RECORD:length Use this option with /FORTRAN to override the default record length of 132 characters for ASCII sequential formatted input and output. The meaningful range for length is from 4 to 4095.

/RUN Use this option to initiate execution of your program if there are no errors in the compilation or the link. This is the default operation. Do not use /RUN with any option that requires a response from the terminal.

/NORUN Use this option to suppress execution of your program. The system performs only the compilation and the link.

/SHOW[:value] Use this option with /FORTRAN to control FORTRAN listing format. The argument *value* represents a code that indicates which listings the compiler is to produce. Table 4–5 summarizes the codes and their meaning.

Use this option with /MACRO to specify any MACRO .LIST directive. Table 4–12 summarizes the valid arguments and their meaning. The *PDP–11 MACRO Language Reference Manual* explains how to use these directives.

/NOSHOW:value Use this option with /MACRO to specify any MACRO .NLIST directive. Table 4–12 summarizes the valid arguments and their meaning. The *PDP–11 MACRO Language Reference Manual* explains how to use these directives.

/STATISTICS Use this option with /FORTRAN to include compilation statistics, such as amount of memory used, amount of time elapsed, and length of the symbol table.

/SWAP Use this option with /FORTRAN to permit the USR (user service routine) to swap over the FORTRAN program in memory. This is the default operation.

/NOSWAP Use this option with FORTRAN to keep the USR resident during execution of a FORTRAN program. This may be necessary if the FORTRAN program uses some of the RT–11 System subroutine library calls (see the *RT–11 Programmer's Reference Manual*). If the program frequently updates or creates a large number of different files, making the USR resident can improve program execution. However, the cost for making the USR resident is 2K words of memory.

/UNITS:n Use this option with /FORTRAN to override the default number of logical units (6) to be open at one time. The maximum value you can specify for *n* is 16.

/VECTORS This option directs FORTRAN to use tables to access multidimensional arrays. This is the default mode of operation.

/NOVECTORS This option directs FORTRAN to use multiplication operations to access multidimensional arrays.

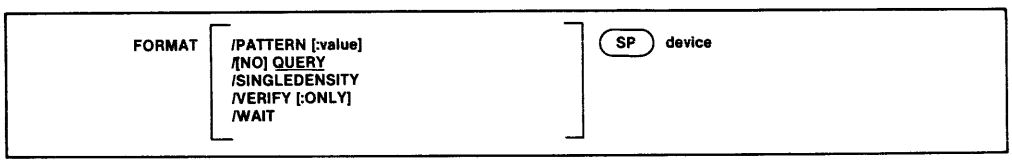
/WARNINGS Use this option to include warning messages in DIBOL or FORTRAN compiler diagnostic error messages. These messages call certain conditions to your attention but do not interfere with the compilation. This is the default operation for DIBOL.

/NOWARNINGS Use this option with /DIBOL to suppress warning messages during compilation. These messages are for your information only; they do not affect the compilation. This is the default operation for FORTRAN.

/WIDE Use this option with /MAP to produce a wide load map listing. Normally, the listing is wide enough for three Global Value columns, which is suitable for a page with 72 or 80 columns. The /WIDE option produces a listing that is six Global Value columns wide, or 132 columns.

FORMAT

The FORMAT command formats disks and diskettes, and verifies any disk, diskette, DECtape, or DECtape II.



In the command syntax described above, *device* represents the storage volume you wish to format and/or verify. Although you can verify any disk or DECtape, the formatting process is valid only for the disks and diskettes listed below.

RX01-RX02
RK05
RP02-RP03
RK06-RK07

When the system formats a volume, it writes headers for each block in the volume. The header of a block contains data the device controller must use to transfer data to and from that block. Using the FORMAT command to format a storage volume makes that volume usable to the RT–11 operating system. Formatting is advisable under the following circumstances:

- when you receive a new RK05 disk from DIGITAL
- when you wish to format an RX02 double density diskette to single density, and vice versa
- when you wish to eliminate bad blocks (though formatting does not guarantee the elimination of every bad block, formatting can reduce the number of bad blocks)

When the system verifies a volume, it writes a 16-bit pattern on each block in the volume, and then reads each pattern. When the system is unable to write and read a pattern, it reports a bad block. The verification process is similar to the bad block scan (see INITIALIZE), except that verification is a data-destructive process. That is, whereas bad block scanning only reads data from each block on a volume, verifying both writes and reads data, destroying any data previously existing on the volume. Because the verification process reads and writes data, it can be more effective than a bad block scan in establishing the validity of data contained in a block. Verifying also makes sure that the previous formatting operation was successful.

NOTE

You can format a diskette (RX01 or RX02) only when you have mounted the diskette in a double density diskette

drive unit (DY). Unless you use the /SINGLEDENSITY option, the system will format both single and double density diskettes in double density format. If you attempt to format a diskette in a single density drive unit (DX), the system will print an error message.

When you format an RK06 or RK07 disk, the system lists the block numbers of all the bad blocks in the manufacturer's bad block table and in the software bad block table.

The options you can use with the FORMAT command follow.

/PATTERN[:value] Use this option with /VERIFY[:ONLY] to specify which 16-bit patterns you want the system to use when it verifies the volume. The optional argument *value* represents an octal integer in the range 0 to 377 that denotes which of eight patterns you want used. The /P:n option in Chapter 18 provides a complete description of the patterns you can specify with /PATTERN[:value]. As the system uses each pattern, it prints at the terminal which pattern it is using.

The command line that follows verifies an RK05 disk with the 16-bit patterns denoted by the value 25.

```
.FORMAT/VERIFY/PATTERN:25  RK0:
RK0:/FORMAT-Are you sure? Y
?FORMAT-I-Formatting complete
PATTERN #5
PATTERN #3
PATTERN #1
?FORMAT-I-Verification complete
```

If you do not supply a value with /PATTERN, the system uses pattern #8.

/QUERY Use this option when you want the system to print a confirmation message before it performs formatting or verification. This is the default setting.

/NOQUERY Use this option if you do not want the system to print a confirmation message before it performs formatting or verification. When you use this option in the FORMAT command line, the system prints only the pattern numbers it uses if it performs verification and the informational messages indicating the formatting or verification is complete. The default setting is /QUERY.

/SINGLEDENSITY Use this option to format an RX02 double density diskette in single density format. The following example uses the /SINGLEDENSITY option to format an RX02 in drive unit 1 as a single density diskette.

```
.FORMAT/SINGLEDENSITY DY1:
DY1:/FORMAT-Are you sure? Y
?FORMAT-I-Formatting complete
```

/VERIFY[:ONLY] Use this option when you want to verify a volume following formatting. Use the optional argument, :ONLY, when you want the

system to only verify a volume. (Note that although you can format only a limited variety of storage volumes, you can verify any disk, diskette, DECtape, or DECtape II.) When you use /VERIFY, the system first formats the specified volume, and then writes a bit pattern to each block on the volume. Next, the system reads each pattern. After the verification process is complete, the system prints at the terminal the block number of each bad block it found.

The example that follows uses /VERIFY to format and verify an RK05 disk in drive unit 2.

```
.FORMAT/VERIFY RK2:
RK2:/FORMAT-Are you sure? Y
?FORMAT-I-Formatting complete
PATTERN #8
?FORMAT-I-Verification complete
```

The next example uses /VERIFY:ONLY to only verify an RX01 diskette in drive unit 0.

```
.FORMAT/VERIFY:ONLY DX0:
DX0:/VERIFY-Are you sure? Y
PATTERN #8
?FORMAT-I-Verification complete
```

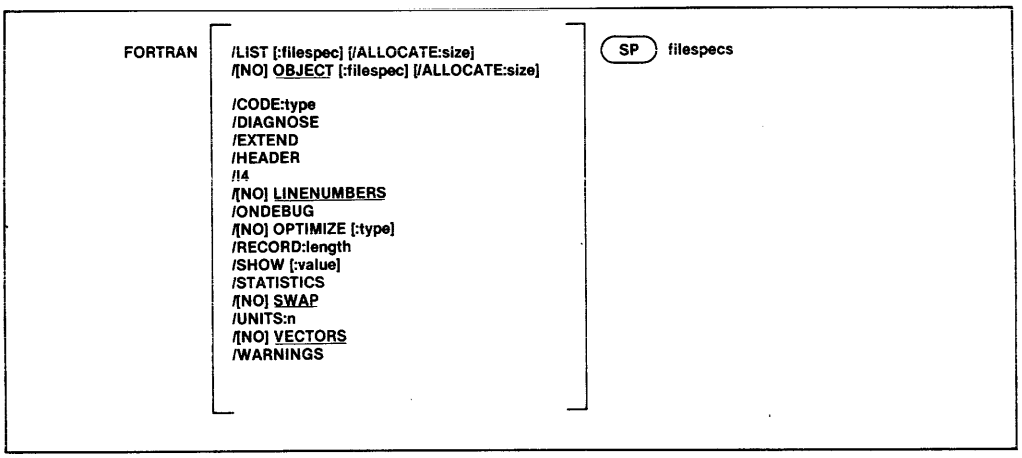
/WAIT Use this option to pause before formatting begins in order to substitute a second volume for the volume you specify in the command line. The /WAIT option is useful for single drive systems. After the system accepts your command line, it pauses while you exchange volumes. Type a Y followed by a carriage return when you have exchanged volumes and are ready for formatting to begin. When formatting completes, the system pauses again while you replace the system volume. Type a Y followed by a carriage return after you have remounted the system volume to terminate the formatting operation.

The following example uses the /WAIT option to format an RK05 disk.

```
.FORMAT/WAIT RK0:
RK0:/FORMAT-Are you sure? Y
Insert volume you wish to format.  CONTINUE(Y/N)?Y
/FORMAT-I-Formatting complete
replace original volume.           CONTINUE(Y/N)?Y
```

FORTRAN

The FORTRAN command invokes the FORTRAN IV compiler to compile one or more source programs.



You can enter the FORTRAN command as one line, or you can rely on the system to prompt you for information. The FORTRAN command prompt is *Files?* for the input specification.

In the command syntax illustrated above, *filespecs* represents one or more files to be included in the compilation. If you omit a file type for an input file, the system assumes .FOR. Output default file types are .LST for listing files and .OBJ for object files. To compile multiple source files into a single object file, separate the files with plus (+) signs in the command line. Unless you specify otherwise, the system creates an object file with the same name as the first input file and gives it an .OBJ file type. To compile multiple files in independent compilations, separate the files with commas (,) in the command line. This generates a corresponding object file for each set of input files.

Language options are position-dependent — that is, they have different meanings depending on where you place them in the command line. Options that qualify a command name apply across the entire command string. Options that follow a file specification apply only to the file (or group of files separated by plus signs) that they follow in the command string.

The *RT–11/RSTS/E FORTRAN IV User's Guide* contains detailed information about using FORTRAN. The following sections describe the options you can use with the FORTRAN command.

/ALLOCATE:size Use this option with /LIST or /OBJECT to reserve space on a device for the output file. The argument *size* represents the number of blocks of space to allocate. The meaningful range for this value is from 1 to 32767. A value of –1 is a special case that creates the largest file possible on the device.

/CODE:type Use this option to produce object code that is designed for a particular hardware configuration. The argument *type* represents a three-letter abbreviation for the type of code to be produced. The legal values are: EAE, EIS, FIS, and THR. See the *RT–11/RSTS/E FORTRAN IV User's Guide* for a complete description of the types of code and their function.

/DIAGNOSE Use this option to help analyze an internal compiler error. /DIAGNOSE expands the crash dump information to include internal compiler tables and buffers. Submit the diagnostic printout to DIGITAL with a software performance report (SPR) form. The information in the listing can help DIGITAL programmers locate the compiler error and correct it.

/EXTEND Use this option to change the right margin for source input lines from column 72 to column 80.

/HEADER This option includes in the printout a list of options that are currently in effect.

/I4 Use this option to allocate two words for the default integer data type (FORTRAN uses one-word integers) so that it takes the same physical space as real variables.

/LINENUMBERS Use this option to include internal sequence numbers in the executable program. These are especially useful in debugging a FORTRAN program. They identify the FORTRAN statements that cause run-time diagnostic error messages. This is the default operation.

/NOLINENUMBERS This option suppresses the generation of internal sequence numbers in the executable program. This produces a smaller program and optimizes execution speed. Use this option to compile only those programs that are already debugged; otherwise the line numbers in FORTRAN error messages are replaced by question marks and the messages are difficult to interpret.

/LIST[:filespec] You must specify this option to produce a FORTRAN compilation listing. The /LIST option has different meanings depending on where you place it in the command line.

The /LIST option produces a listing on the line printer when /LIST follows the command name. For example, the following command line produces a line printer listing after compiling a FORTRAN source file:

```
.FORTRAN/LIST MYPROG<RET>
```

When the /LIST option follows the file specification, it produces a listing file. For example, the following command line produces the listing file DK:MYPROG.LST after compiling a FORTRAN source file:

```
.FORTRAN MYPROG/LIST<RET>
```

If you specify /LIST without a file specification in the list of options that immediately follows the command name, the FORTRAN compiler generates a listing that prints on the line printer. If you follow /LIST with a device name, the system creates a listing file on that device. If the device is a file-

structured device, the system stores the listing file on that device, assigning it the same name as the input file with a .LST file type. The following command produces a listing on the terminal:

```
.FORTRAN/LIST:TT: A
```

The next command creates a listing file called A.LST on RK3:.

```
.FORTRAN/LIST:RK3: A
```

If the /LIST option contains a name and file type to override the default of .LST, the system generates a listing file with that name. The following command, for example, compiles A.FOR and B.FOR together, producing files A.OBJ and FILE1.OUT on device DK:.

```
.FORTRAN/LIST:FILE1.OUT A+B
```

Another way to specify /LIST is to type it after the file specification to which it applies. To produce a listing file with the same name as a particular input file, you can use a command similar to this one:

```
.FORTRAN A+B/LIST:RK3:
```

The above command compiles A.FOR and B.FOR together, producing files DK:A.OBJ and RK3:B.LST. If you specify a file name on a /LIST option following a file specification in the command line, it has the same meaning as when it follows the command. The following two commands have the same results:

```
.FORTRAN A/LIST:B
```

```
.FORTRAN/LIST:B A
```

Both the above commands generate A.OBJ and B.LST as output files.

Remember that file options apply only to the file (or group of files that are separated by plus signs) that they follow in the command string. For example:

```
.FORTRAN A/LIST,B
```

This command compiles A.FOR, producing A.OBJ and A.LST. It also compiles B.FOR, producing B.OBJ. However, it does not produce any listing file for the compilation of B.FOR.

/OBJECT[:filespec] Use this option to specify a file name or device for the object file. Because FORTRAN creates object files by default, the following two commands have the same meaning:

```
.FORTRAN A
```

```
.FORTRAN/OBJECT A
```

Both commands compile A.FOR and produce A.OBJ as output. The /OBJECT option functions like the /LIST option; it can be either a command or a file qualifier.

As a command option, /OBJECT applies across the entire command string. The following command, for example, compiles A.FOR and B.FOR separately, creating object files A.OBJ and B.OBJ on RK1:.

```
.FORTRAN/OBJECT:RK1: A,B
```

Use /OBJECT as a file option to create an object file with a specific name or destination. The following command compiles A.FOR and B.FOR together, creating files B.LST and B.OBJ.

```
.FORTRAN A+B/LIST/OBJECT
```

/NOOBJECT Use this option to suppress creation of an object file. As a command option, /NOOBJECT suppresses all object files; as a file option, it suppresses only the object file produced by compilation of the related input files. In this command, for example, the system compiles A.FOR and B.FOR together, producing files A.OBJ and B.LST. It also compiles C.FOR and produces C.LST, but does not produce C.OBJ.

```
.FORTRAN A+B/LIST,C/NOOBJECT/LIST
```

/ONDEBUG Use this option to include debug lines (those that have a D in column one) in the compilation. You do not, therefore, have to edit the file to include these lines in the compilation or to logically remove them. This option is useful in debugging a program. You can include messages, flags, and conditional branches to help you trace program execution and find an error.

/OPTIMIZE:type Use this option to enable certain options that optimize object code for various conditions. The argument *type* represents the three-letter code for the type of optimization to be enabled. Table 4–4 summarizes the codes and their meaning.

Table 4–4: Optimization Codes

Code	Meaning
BND	Global register bindings for inline code generation
CSE	Common subexpression elimination
SPD	Optimization for speed of execution, as opposed to minimal program size
STR	Strength reduction optimization

This option is not available with version 2.5 of the FORTRAN compiler.

/NOOPTIMIZE:type Use this option to disable certain options that optimize object code for various conditions. The argument *type* represents the three-letter code for the type of optimization to be disabled. Table 4–4 summarizes the codes and their meaning. This option is not available with version 2.5 of the FORTRAN compiler.

/RECORD:length Use this option to override the default record length of, usually 132, characters for ASCII, sequentially formatted input and output. The meaningful range for length is from 4 to 4095.

/SHOW[:value] Use this option to control FORTRAN listing output. The argument *value* represents a code that indicates which listings the compiler is to produce. Table 4–5 summarizes the codes and their meaning. You can combine options by specifying the sum of their numeric codes. For example:

/SHOW:7

or

/SHOW:ALL

The two options shown above have the same meaning. If you specify no code, the default value is 3, a combination of SRC and MAP.

Table 4–5: FORTRAN Listing Codes

Code	Listing Content
0	Diagnostics only
1 or SRC	Source program and diagnostics
2 or MAP	Storage map and diagnostics
3	Diagnostics, source program, and storage map
4 or COD	Generated code and diagnostics
7 or ALL	Diagnostics, source program, storage map, and generated code

/STATISTICS Use this option to include compilation statistics in the listing, such as amount of memory used, amount of time elapsed, and length of the symbol table.

/SWAP Use this option to permit the USR (user service routine) to swap over the FORTRAN program in memory. This is the default operation.

/NOSWAP This option keeps the USR resident during execution of a FORTRAN program. This may be necessary if the FORTRAN program uses some of the RT–11 system subroutine library calls (see Chapter 4 of the *RT–11 Programmer's Reference Manual*). If the program frequently updates or creates a large number of different files, making the USR resident can improve program execution. However, the cost for making the USR resident is 2K words of memory.

/UNITS:n Use this option to override the default number of logical units (6) to be open at one time. The maximum value you can specify for *n* is 16.

/VECTORS This option directs FORTRAN to use tables to access multidimensional arrays. This is the default mode of operation.

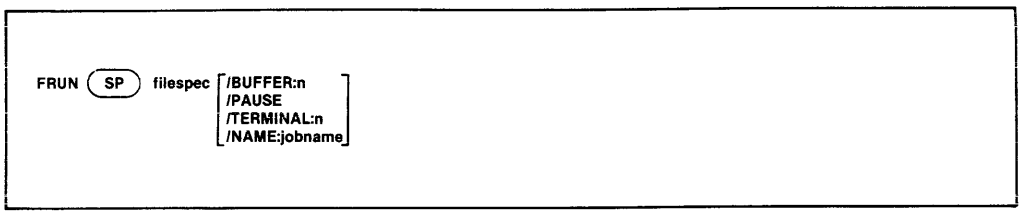
/NOVECTORS This option directs FORTRAN to use multiplication operations to access multidimensional arrays.

/WARNINGS Use this option to include warning messages in FORTRAN compiler diagnostic error messages. These messages call certain conditions to your attention, but do not interfere with the compilation. A warning mes-

sage prints, for example, if you change an index within a DO loop, or if you specify a variable name longer than six characters.

/NOWARNINGS Use this option to exclude warning messages in FORTRAN compiler diagnostic error messages. This is the default setting.

The FRUN command initiates foreground jobs. The default file type is .REL.



In the command syntax illustrated above, *filespec* represents the program to execute. Because this command runs a foreground job, it is valid for the FB and XM monitors only.

If a foreground job is active when you issue the FRUN command, an error message prints on the terminal. You can run only one foreground job at a time. If a terminated foreground job is occupying memory, the system reclaims that region for your program. Then, if the system finds your program and if your program fits in the available memory, execution begins.

The following sections describe the options you can use with FRUN. Note that the option must follow the file specification in the command line.

Note that you can use the FRUN command to run a virtual foreground job, and that you can use FRUN to run a virtual .SAV file in the foreground under the XM monitor.

/BUFFER:n Use this option to reserve more space in memory than the actual program size. The argument *n* represents, in octal, the number of words of memory to allocate. You must use this option to execute a FORTRAN foreground job. If you use /BUFFER for a virtual job linked with the /V option (or /XM), the system ignores /BUFFER because it has already provided a buffer in extended memory.

The following formula determines the space needed to run a FORTRAN program as a foreground job.

$$n = [1/2[504 + (33*N) + (R-136) + A*512)]]$$

where:

- A represents the number of files open at one time. If you are using double buffering, multiply A by 2.
- N represents the number of channels (logical unit numbers)
- R represents the maximum record length. The default is 136 characters.

This formula must be modified for certain System Subroutine Library (SYSLIB) functions.

The IQSET function requires the formula to include additional space for queue elements (qcount) as follows:

$$n = [1/2[504 + (33*N) + (R-136) + A*512]] + [10*qcount]$$

The ICDFN function requires the formula to include additional space for the integer number of channels (num) as follows:

$$n = [1/2[504 + (33*N) + (R-136) + A*512]] + [6*num]$$

The INTSET function requires the formula to include additional space for the number of INTSET calls issued in the program as follows:

$$n = [1/2[504 + (33*N) + (R-136) + A*512]] + [25*INTSET]$$

Any functions, including INTSET, that invoke completion routines must include 64(decimal) words plus the number of words needed to allocate the second record buffer (default is 68(decimal) words). The length of the record buffer is controlled by the /RECORD option to the FORTRAN compiler. If the /RECORD option is not used, the allocation in the formula must be 136(decimal) bytes, or the length that was set at FORTRAN installation time. This modifies the formula as follows:

$$n = [1/2[504 + (33*N) + (R-136) + A*512]] + [64 + R/2]$$

If the /BUFFER option does not allocate enough space in the foreground on the initial call to a completion routine, the following message appears:

```
?ERR 0, NON-FORTRAN error call
```

This message also appears if there is not enough free memory for the background job or if a completion routine in the single-job monitor is activated during another completion routine. In the latter case, the job aborts; you should use the FB monitor to run multiple active completion routines.

/PAUSE Use this option to help you debug a program. When you type the carriage return at the end of the command string, the system prints the load address of your program and waits. You can examine or modify the program (by using ODT, described in Chapter 21) before starting execution. You must use the RESUME command to start the foreground job. The following command loads the program DEMOSP.REL, prints the load address, and waits for a RESUME command to begin execution.

```
•FRUN DEMOSP/P
Loaded at 127276
•RESUME
```

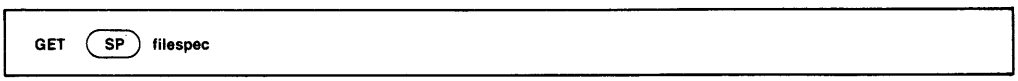
/TERMINAL:n This option is meaningful only in a multi-terminal system. Use it to assign a terminal to interact with the foreground job. The argument *n* represents a terminal logical unit number. If you do not use this option, the foreground job shares the console terminal with the background job. By assigning a different terminal to interact with the foreground job, you eliminate the need for the foreground and background jobs to share the console terminal. Note that the original console terminal still interacts with

the background job and with the keyboard monitor, unless you use the SET TT:CONSOL command to change this.

/NAME:name Use this option to assign a logical name to the foreground job. This option is valid only on a monitor that has system job support, a special feature enabled by the system generation process.

GET

The GET command loads a memory image file into memory.



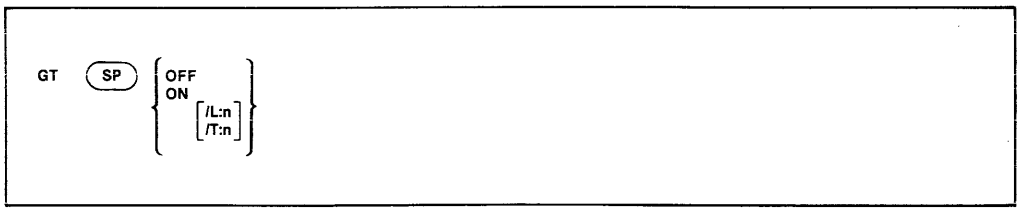
In the command syntax shown above, *filespec* represents the memory image file to be loaded. The default file type is .SAV. Note that magtape and cassette are not block-replaceable devices and therefore are not permitted with the GET command. Use the GET command for a background job only. You cannot use GET on a virtual program that executes under the XM monitor. The GET command is useful when you need to modify or debug a program. You can use GET with the Base, Deposit, Examine, and START commands to test changes. Use the SAVE command to make these changes permanent. You can combine programs by issuing multiple GET commands, as the following example shows. This example loads a program, DEMOSP.SAV, loads ODT.SAV (on-line debugging technique, described in Chapter 21), and starts the program using the address of ODT's entry point.

```
.GET DEMOSP

.GET ODT

.START

ODT V01.04
*
```

If more than one program requires the same locations in memory, the program you load later overlays the previous program. Note that you cannot use GET to load overlay segments of a program; it can load only the root. If the file you need to load resides on a device other than the system device, the system automatically loads that device handler into memory when you issue the GET command. This prevents problems that occur if you use the START command and your program is overlaid.

GT

The GT command enables or disables the VT11 or VS60 display hardware.



When you issue the GT OFF command, you disable the display hardware. The printing console terminal then becomes the device that prints output from the system.

When you issue the GT ON command, the display screen replaces the printing console terminal. The display screen offers some advantages over the printing terminal: (1) it is quieter than a printing terminal, (2) it is faster than a printing terminal, (3) it does not require a supply of paper, and (4) it is the device for which EDIT's immediate mode is intended. The display screen can speed up the editing process (see Chapter 5 for information on how to use the text editor). You can use CTRL/A, CTRL/S, CTRL/E, and CTRL/Q to control scrolling. These commands are explained in Chapter 3.

Note that RT–11 does not permit you to use display hardware (with GT ON) if you have multi-terminal support (enabled by a user-generated monitor) or if you have an 8K configuration. You cannot use GT ON or GT OFF when a foreground or system job is active; this causes the system to print an error message. Issue the GT ON command before you begin execution of the foreground job. ODT (on-line debugging technique, described in Chapter 21) is the only system program that cannot use the display screen. Its output always appears on the console terminal. You can use VDT, a variant of ODT, because it can interact with the display hardware.

NOTE

If an indirect command file issues a GT ON command, part of the command may echo on the terminal and the rest may echo on the graphics screen. Also, if you type the GT ON command, followed by CTRL/E, the initial line on the terminal overprints when you type GT OFF.

The following options control the number of lines that appear on the screen and position the first line vertically.

/L:n Use this option to change the number of lines of text that display on the screen. Table 4–6 shows the valid range for the argument *n* in decimal. If you do not use this option the system determines the screen size and automatically assigns the largest valid value.

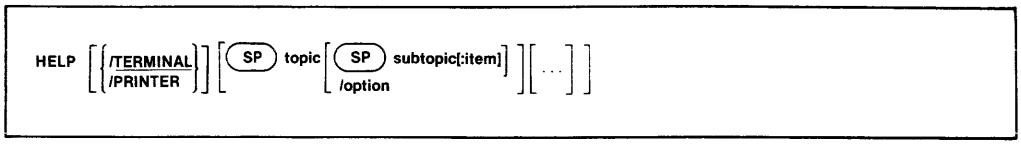
/T:n Use this option to change the top position of the scroll display. Table 4–6 shows the valid range for the argument *n* in decimal. If you do not use this option, the system determines the screen size and automatically assigns the largest valid value.

Table 4–6: Display Screen Values

Screen Size	Lines	Top Position
12-inch	1–31	1–744
17-inch	1–40	1–1000 (or larger)

HELP

The HELP command lists information related to RT–11 commands to help you remember command syntax, options, and so on, when you are at the console.



In the command syntax shown above, *topic* represents a subject about which you need information. In the help file supplied with RT–11, the topics are the keyboard monitor commands. The *subtopic* represents a category within a topic. In the RT–11 help file, the subtopics are SYNTAX, SEMANTICS, OPTIONS, and EXAMPLES. The *item* represents one member of the subtopic group. You can specify more than one item in the command line if you separate the items by colons (:). If you type HELP followed by a carriage return, the system lists information on the HELP command.

The HELP command permits you to access the HELP text file. The help file distributed with RT–11 contains information about the keyboard monitor commands and how to use them. However, the concept of the help file is a general one. That is, you can create your own help file to supply quick reference material on any subject. For information on how to change the HELP text file, see the *RT–11 Installation and System Generation Guide*. There are only two options you can use with the HELP command. They are /PRINTER and /TERMINAL.

/PRINTER Use this option to list information on the line printer.

/TERMINAL This option lists information on the console terminal. This is the default operation.

The following examples all make use of the standard RT–11 help file.

The following command lists all the topics for which assistance is available.

```
.HELP *

APL         Invokes the APL language interpreter
ASSIGN      Associates a logical device name with a physical device
B           Sets a relocation base
  .
  .
  .
```

The next command lists all the information about the DATE command.

```
.HELP DATE

DATE      Sets or displays the current system date

  SYNTAX
        DATE[ dd-mmm-yy]

  SEMANTICS
        All numeric values are decimal;  mmm is the first three
        characters of the name of the month

  OPTIONS
        None

  EXAMPLES
        DATE 12-MAY-79
```

The next command lists all the options that are valid with the DIRECTORY command.

```
.HELP DIRECTORY OPTIONS

  OPTIONS
    ALLOCATE:size
        Use with /OUTPUT to reserve space for the output  listing
        file
    ALPHABETIZE
        Sorts the directory in alphabetical order  by  file  name
        and type
      .
      .
      .
```

The next command lists information about the /BRIEF option for the DIRECTORY command.

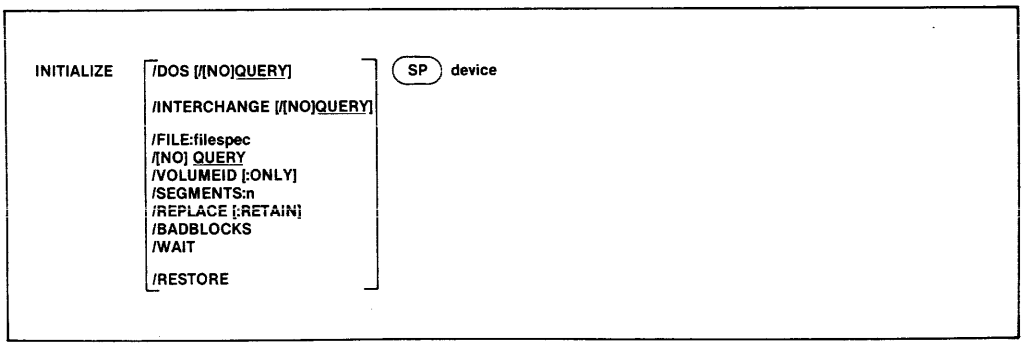
```
.HELP DIRECTORY OPTIONS:BRIEF

    BRIEF
        Lists only file names and file types of files;   same  as
        /FAST
```

The following command lists information about the DIRECTORY command options that begin with B.

```
.HELP DIRECTORY/B

    BADBLOCKS

        Scans the device for bad blocks  and  types  their  octal
        number
    BEFORE[date]
        Lists the files created before the date you specify
    BEGIN
        Lists the directory, starting with the file you specify
    BLOCKS
        Lists the starting block numbers of the files
    BRIEF
        Lists only file names and file types of files;   same  as
        /FAST
```

INITIALIZE

Use the INITIALIZE command to clear and initialize a device directory.



In the command syntax illustrated above, *device* represents the volume you need to initialize. The initialize operation must always be the first operation you perform on a new volume after you receive it, formatted, from the manufacturer. If the volume is not formatted, use the FORMAT command (see the FORMAT command description) to format the volume. The INITIALIZE command destroys any data that may already exist on a device. After you use the INITIALIZE command, there are no files in the directory. If you use the INITIALIZE command with no options, the system simply initializes the device directory. You can enter the INITIALIZE command as one line, or you can rely on the system to prompt you for the name of the device with *Device?*. The following sections describe the options you can use with INITIALIZE and give some examples of their use.

The default number of directory segments for RT–11 directory structured volumes is listed in Table 4–7. If any default is too small for your needs, see the *RT–11 Installation and System Generation Guide* for details on changing this default directory size.

If the volume you are initializing has protected files, the system always prints a confirmation message as in the following example.

```
.INIT RK0:
RK0:/Initialize; Are you sure? Y <RET>
Volume contains protected files; Are you sure? Y <RET>
```

/BADBLOCKS[:RET] Use this option to scan a volume (disk or DECtape) for bad blocks and write .BAD files over them. For each bad block the system encounters on the volume, it creates a file called FILE.BAD to cover it. After the volume is initialized and the scan completed, the directory consists of only FILE.BAD entries to cover the bad blocks. This procedure ensures that the system will not attempt to access these bad blocks during routine operations. If the system finds a bad block in either the boot block or the volume directory, it prints an error message and the volume is not usable. DIGITAL recommends that you use the DIRECTORY/BADBLOCKS command after using the INITIALIZE/BADBLOCKS command so that you can find out

where the bad blocks are, if any. The following command initializes volume RK1: and scans for bad blocks.

```
.INITIALIZE/BADBLOCKS RK1:
RK1:/Initialize;  Are your sure?  Y
```

If you use /BADBLOCKS:RET, the system will retain through initialization all files with a .BAD file type that it finds on the volume, giving them the name FILE.BAD. The system does not do a bad block scan. The advantage in using :RET is that initializing takes less time. Note that some volumes support bad block replacement; DIGITAL recommends you use the /REPLACE[:RET] option instead of /BADBLOCKS[:RET] for these volumes when scanning for bad blocks.

/DOS Use this option to initialize a DECtape for DOS–11 format.

/FILE:filespec Use this option to initialize a magtape and create a bootable tape. For *filespec,* substitute *dev:MBOOT.BOT*. This file is distributed with RT–11 for this purpose only. Consult the *RT–11 Installation and System Generation Guide* for more information. The following example creates a bootable magtape.

```
.INITIALIZE/FILE:MBOOT.BOT  MT0:
```

/INTERCHANGE Use this option to initialize a diskette for interchange format. The following example initializes DX1: in interchange format.

```
.INITIALIZE/INTERCHANGE DX1:
DX1:/Init  Are you sure?  Y
```

/QUERY This option prompts you for confirmation before it initializes a device. Respond by typing a Y followed by a carriage return to initiate execution of the command. The system interprets a response beginning with any other character to mean NO. /QUERY is the default operation.

/NOQUERY Use this option to suppress the confirmation message the system prints before it proceeds with the initialization.

/REPLACE[:RET] If you have an RK06, RK07, RL01, or RL02 disk, use this option to scan a disk for bad blocks. If the system finds any bad blocks, it creates a replacement table so that routine operations access good blocks instead of bad ones. Thus, the disk appears to have only good blocks. Note, though, that accessing this replacement table slows response time for routine input and output operations. With /REPLACE, you have the option of deciding which bad blocks you want replaced if there is a replacement table overflow. The RK06s and RK07s support up to 32 bad blocks in the replacement table; the RL01s and RL02s support up to 10.

With an RK06 or RK07 disk, the system can replace only those bad blocks that generate a bad sector error (BSE). Of the blocks the system cannot replace, the system can report a bad block as being hard or soft. If you use /VERIFY with /REPLACE and the system still cannot use the block, the system reports a hard bad block. If the system can use the block, it reports a soft bad block.

With an RL01 or RL02 disk, the system can replace any kind of bad block.

When you use /REPLACE, the system prints a list of replaceable bad blocks as in the following sample:

```
.INITIALIZE/REPLACE DL0:

    Block          Type
030722  12754.  Replaceable
115046  39462.  Replaceable
133617  46991.  Replaceable
136175  48253.  Replaceable
136277  48319.  Replaceable
136401  48385.  Replaceable
140405  49413.  Replaceable
146252  52394.  Replaceable
DUP-I-Bad blocks detected 8.
```

If there is a replacement table overflow, the system prompts you to indicate which blocks you want replaced as follows:

```
?DUP-W-Replacement table overflow
Type <RET>, 0, or nnnnnn (<RET>)
Replace block #
```

nnnnnn represents the octal number of the block you want the system to replace.

After you enter a block number, the system responds by repeating the *Replace block #* prompt. If you type a 0 at any time you do not want any more blocks replaced, prompting ends and any blocks not placed in the replacement table are marked as FILE.BAD.

If you enter a carriage return at any time, the system places all bad blocks you have not entered into the replacement table, starting with the first on the disk, until the table is full. The system assigns the name FILE.BAD to any remaining bad blocks and prompting ends.

If you use /NOQUERY with /REPLACE, and there is a replacement table overflow, the effect will be as if you had entered a carriage return in response to the first *Replace block #* prompt.

If you use :RET with /REPLACE, the system initializes the volume and retains the bad block replacement table (and FILE.BAD files) created by the previous /REPLACE command.

Note that if the monitor file resides on a block that contains a bad sector error (BSE) and you are doing bad block replacement, a boot error results when you attempt to bootstrap the system. In this case, move the monitor. Use the DIRECTORY/BADBLOCKS/FILES command to determine which files reside on bad blocks.

/RESTORE Use this option to *uninitialize* a volume. That is, you can use this option to restore the directory and files that were present on the volume prior to the previous initialization. You can use /RESTORE only if no files have been transferred to the volume since the last time it was initialized.

The /RESTORE option does not restore the boot blocks; so if you use /RESTORE to restore a previously bootable volume, use the COPY/BOOT command to make the volume bootable again.

/SEGMENTS:n Use this option if you need to initialize a disk and also change the number of directory segments. The number of segments in the directory establishes the number of files that can be stored on a device. The system allows a maximum of 72 files per directory segment, and 31 directory segments per device. The argument *n* represents the number of directory segments. The valid range for *n* is from 1 to 31 (decimal). Table 4–7 shows the default values of *n* for standard RT–11 devices.

Table 4–7: Default Directory Sizes

Device	Size (decimal) of Directory in Segments
RK	16
DD	1
DT	1
RF	4
DS	4
DP	31
DX	1
DM	31
DY	4
DL	16
PD	1

/VOLUMEID[:ONLY] Use /VOLUMEID to write a volume identification on a device when you initialize it. This identification consists of a volume ID (up to 12 characters long for a block-replaceable device, up to 6 characters long for magtape) and an owner name (up to 12 characters long for a block-replaceable device, up to 10 characters long for magtape). The following example initializes device RK1: and writes a volume identification on it.

```
.INITIALIZE/VOLUMEID RK1:
RK1:/Initialize; Are you sure? Y
 Volume ID? FORTRAN VOL
 Owner?     Marcy
```

Use /VOLUMEID:ONLY to write a new volume identification on a device without reinitializing the device. You cannot change the volume ID of a magtape or cassette without initializing the entire tape.

/WAIT The /WAIT option is useful if you have a single-disk system. When you use this option to initialize a volume, the system begins the procedure but then pauses and waits for you to mount the volume you want to initialize. When the system pauses, it prints the following prompt at the terminal:

```
Mount input volume in <device>;  Continue?
```

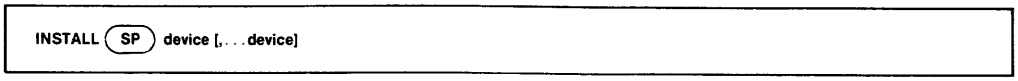
<device> is the name of the device into which you mount the volume to be initialized. After you have mounted the input volume, type Y followed by a carriage return. After the system completes the initialization process, it prints the following message prompting you to mount the system volume:

```
Mount system volume in <device>;  Continue?
```

After you mount the system volume, type Y followed by a carriage return. When you use /WAIT, make sure that DUP is on the system volume.

INSTALL

The INSTALL command installs the device you specify into the system.



In the command syntax shown above, *device* represents the name of the device to be installed. The INSTALL command accepts no options. It allows you to install into the system tables a device that was not installed into the system when it was bootstrapped. (A device handler must exist in the system tables before you can use that device.) The device occupies the first available device slot. Using the INSTALL command does not change the monitor disk image; it only modifies the system tables of the monitor that is currently in memory.

You can enter the command on one line, or you can rely on the system to prompt you for information. The INSTALL command prompt is *Device?*.

When you specify a device name, the system searches the system device for the corresponding device handler file. For SJ and FB systems, if LP: is to be installed, the INSTALL command searches for the file SY:LP.SYS. For XM systems, INSTALL searches for SY:LPX.SYS. The INSTALL command does not allow a device handler built for a different configuration of the system to be installed in a given system. Note that you cannot install the device names SY, DK, or BA.

To permanently install a device, include the INSTALL command in the standard, system startup indirect command file. This file is invoked automatically when you boot the system. The INSTALL command also allows you to configure a special system for a single session without having to reconfigure to revert to the standard device configuration. If there are no free device slots (use the SHOW DEVICES command to ascertain this), you must remove an existing device (with the REMOVE command) before you can install a new device.

The following command installs the card reader into the system tables from the file CR.SYS. (The colon (:) that follows the device handler name is optional.)

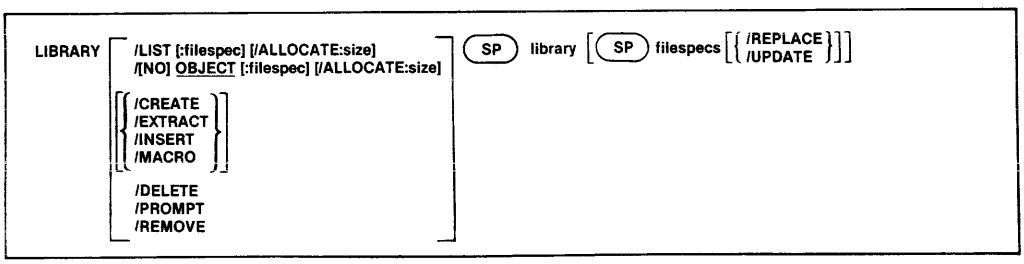
```
.INSTALL CR:
```

The next example installs the line printer, the card reader, and DECtape.

```
.INSTALL LP:,CR:,DT:
```

LIBRARY

The LIBRARY command lets you create, update, modify, list, and maintain library files.



In the command syntax illustrated above, *library* represents the library file name, and *filespecs* represents the input module file names. Separate the library file specification from the module file specifications with a space. Separate the module file specifications with commas. The system uses .LST as the default file type for library directory listing files. It uses .OBJ as the default file type for object libraries and object input files, and .MAC for macro libraries and macro input files. Object libraries contain machine-level object modules, and macro libraries contain MACRO source modules. You cannot combine object modules with MACRO modules. The default operation, if you do not specify an option, is /INSERT. If you do not specify a library file in the command line, the system prompts *Library?*. If you specify /CREATE, /INSERT, or /MACRO and omit the module file specification, the system prompts *Files?*. If you specify /EXTRACT, the system prompts *File?*. Note that no other option causes the *File?* or *Files?* prompt.

The LIBRARY command can perform all the functions listed above on object library files. It can also create macro library files for use with the MACRO–11 assembler. A library file is a direct-access file (a file that has a directory) that contains one or more modules of the same type. The system organizes library files so the linker and MACRO–11 assembler can access them rapidly. Each library is a file that contains a library header, library directory, and one or more object modules. The object modules in a library file can be routines that are repeatedly used in a program, routines that are used by more than one program, or routines that are related and simply gathered together for convenience. An example of a typical object library file is the default system library, SYSLIB.OBJ, used by the linker. An example of a macro library file is SYSMAC.SML.

You access object modules in a library file by making calls or references to their global symbols; you link the object modules with the program that uses them by using the LINK command to produce a single executable module. Each input file for an object library consists of one or more object modules, and is stored on a device under a specific file name and file type. Once you insert an object module into a library file, you no longer reference the module by the file name of which it was a part; reference it by its individual module name. For example, the input file FORT.OBJ may exist on DT2: and can

contain an object module called ABC. Once you insert the module into a library, reference only ABC, and not FORT.OBJ.

The input files normally do not contain main programs but only subprograms, functions, and subroutines. The library file must never contain a FORTRAN BLOCK DATA subprogram because there is no undefined global symbol to cause the linker to load it automatically.

The following sections describe the LIBRARY command options and explain how to use them. The last section under this command describes the LIBRARY prompting sequence and order of execution for commands that combine two or more LIBRARY options. Chapter 12 contains more detailed information on object and macro libraries.

/ALLOCATE:size Use this option only with /LIST or /OBJECT to reserve space on the device for the output file. The value *size* represents the number of blocks of space to allocate. The meaningful range for this value is from 1 to 32767. A value of –1 is a special case that allocates the largest area available on the device.

The following example uses /ALLOCATE to create the object library MYLIB.OBJ from the object library MYFILE.OBJ. The argument, –1, is specified with /ALLOCATE.

```
LIBRARY/OBJECT:MYLIB/ALLOCATE:-1 MYFILE
```

/CREATE Use this option by itself to create an object library. Specify a library name followed by the file specifications for the modules that are to be included in that library. The following command creates a library called NEWLIB.OBJ from the modules contained in files FIRST.OBJ and SECOND.OBJ.

```
.LIBRARY/CREATE NEWLIB FIRST,SECOND
```

/DELETE Use this option to delete an object module and all its associated global symbols from the library. Specify the library name in the command line. The system prompts you for the names of the modules to delete. The prompt is:

```
Module name?
```

Respond with the name of a module. (Be sure to specify a module name and not a global name.) Follow each module name with a carriage return. Enter a carriage return on a line by itself to terminate the list of module names. The following example deletes modules SGN and TAN from the library called NEWLIB.OBJ.

```
.LIBRARY/DELETE NEWLIB
Module name? SGN
Module name? TAN
Module name?
```

/EXTRACT Use this option to extract an object module from a library and store it in a file with the same name as the module and a file type of .OBJ.

You cannot combine this option with any other option. The system prompts you for the name of the object module to be extracted. The prompt is:

```
Global?
```

If you specify a global name, the system extracts the entire module of which that global is a part. Follow each global name with a carriage return. Enter a carriage return on a line by itself to terminate the list of global symbols. The following example shows how to extract the module ATAN from the library called NEWLIB.OBJ and store it in file ATAN.OBJ on DX1:.

```
.LIBRARY/EXTRACT
Library? NEWLIB
File   ? DX1:ATAN
Global ? ATAN
Global ?
```

/INSERT Use this option to insert an object module into an existing library. Although you can insert object modules that have duplicate names, this practice is not recommended because of the difficulty involved in replacing or updating these modules. Note that /INSERT is the default operation. If you do not specify any option, insertion takes place. The following example inserts the modules contained in the files THIRD.OBJ and FOURTH.OBJ into the library called OLDLIB.OBJ.

```
.LIBRARY/INSERT OLDLIB THIRD,FOURTH
```

/LIST[:filespec] Use this option to obtain a directory listing of an object library. The following example obtains a directory listing of OLDLIB.OBJ on the terminal (the line printer is the default device).

```
.LIBRARY/LIST:TT:  OLDLIB
```

The directory listing prints global symbol names. A plus sign (+) in the module column indicates a continued line. See Section 12.2.7 for a procedure to include module names in the directory listing.

You can also use /LIST with other options (except /MACRO) to obtain a directory listing of an object library after you create or modify it. The following command, for example, inserts the modules contained in the files THIRD.OBJ and FOURTH.OBJ into the library called OLDLIB.OBJ; it then prints a directory listing of the library on the terminal.

```
.LIBRARY/INSERT/LIST:TT:  OLDLIB THIRD,FOURTH
```

You cannot obtain a directory listing of a macro library.

Make sure when you use /LIST with LIBRARY that you use it on the command side of the command string, and not after the file specification.

/MACRO Use this option to create a macro library. Note that this is the only valid function for a macro library. You can create a macro library, but you cannot list or modify it. To update a macro library, simply edit the ASCII text file and then reprocess the file with the LIBRARY/MACRO com-

mand. The following example creates a macro library called NEWLIB.MAC from the ASCII input file SYSMAC.MAC.

```
•LIBRARY/MACRO/CREATE NEWLIB SYSMAC
```

When you use /MACRO with LIBRARY, use it on the command side of the command string, and not after the file specification.

/OBJECT[:filespec] The system creates object library files by default as a result of executing a LIBRARY command. When you modify an existing library, the system actually makes the changes to the library you specify, thus creating a new, updated library that it stores under the same name as the original library. Use this option to give a new name to the updated library file and preserve the original library. The following example creates a library called NEWLIB.OBJ, which consists of the library OLDLIB.OBJ plus the modules that are contained in files THIRD.OBJ and FOURTH.OBJ.

```
•LIBRARY/INSERT/OBJECT:NEWLIB OLDLIB THIRD,FOURTH
```

/NOOBJECT Use this option to suppress the creation of a new object library as a result of a LIBRARY command.

/PROMPT Use this option to specify more than one line of input file specifications in a LIBRARY command. This option is valid with all other library functions except the /EXTRACT option. You must specify // as the last input in order to terminate the input list. Note that the file specifications you enter after typing the /PROMPT option must conform to Command String Interpreter conventions. The following example creates a macro library called MACLIB.MAC from seven input files.

```
•LIBRARY/MACRO/PROMPT MACLIB A, B, C, D
*E,F,G
*//
```

/REMOVE This option permits you to delete a specific global symbol from a library file's directory. Since globals are deleted only from the directory (and not from the object module itself), all the globals that were previously deleted are restored whenever you update that library, unless you use /REMOVE again to delete them. This feature lets you recover a library if you have inadvertently deleted the wrong global. The system prompts you for the names of the global symbols to remove. The prompt is:

```
Global?
```

Respond with the name of a global symbol to be removed. Follow each global symbol with a carriage return. Enter a carriage return on a line by itself to terminate the list of global symbols. The following example deletes the globals GA, GB, GC, and GD from the library OLDLIB.OBJ.

```
•LIBRARY/REMOVE OLDLIB
Global? GA
Global? GB
Global? GC
Global? GD
Global?
```

/REPLACE Use this option to replace modules in an existing object library with modules of the same name contained in the files you specify. The following example replaces a module called SQRT in the library MATHLB.OBJ with a new module, also called SQRT, from the file called MFUNCT.OBJ.

```
.LIBRARY MATHLB MFUNCT/REPLACE
```

Note that the /REPLACE option must follow each file specification that contains a module to be inserted into the library. Note also that you can use /REPLACE only with a module(s), and never a library file(s).

/UPDATE This option combines the functions of /INSERT and /REPLACE. Specify it after each file specification to which it applies. If the modules in the input file already exist in the library, the system replaces those library modules. If the modules in the input file do not exist in the library, the system inserts them. The following example updates the library OLDLIB.OBJ.

```
.LIBRARY OLDLIB FIRST/UPDATE,SECOND/UPDATE
```

Note that the /UPDATE option must follow each file specification to which it applies, and that you can use this option only with modules, not files.

You can combine the LIBRARY options with the exceptions of /EXTRACT and /MACRO, which you cannot combine with most of the other functions. Table 4–8 lists the sequence in which the system executes the LIBRARY options and prompts you for additional information.

Table 4–8: Execution and Prompting Sequence of LIBRARY Options

Option	Prompt
/CREATE /DELETE /REMOVE /UPDATE /REPLACE /INSERT /LIST	Module name? Global?

The following example combines several options.

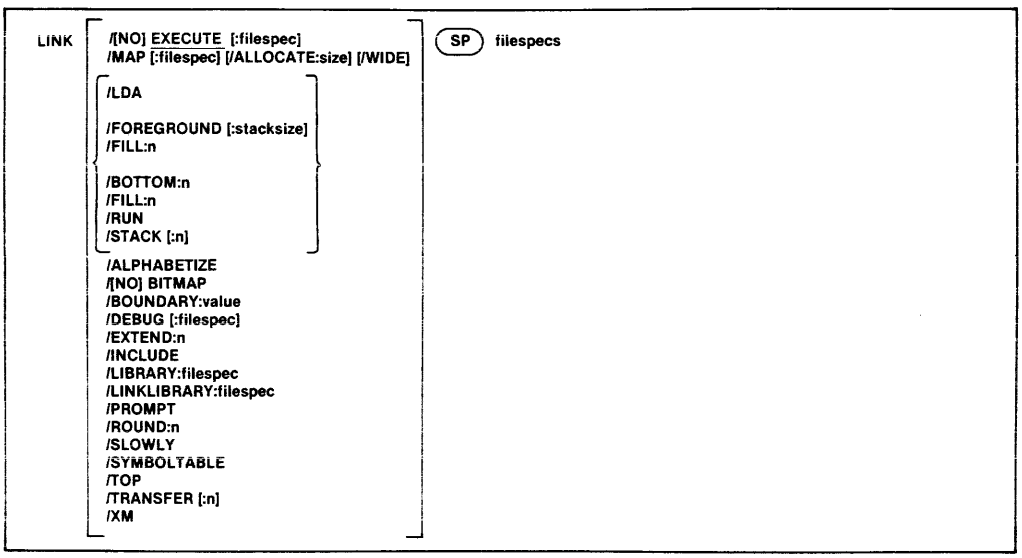
```
LIBRARY/LIST:TT:/REMOVE/INSERT NEWLIB LIB2/REPLACE,LIB3
Global? SQRT
Global?
RT-11 LIBRARIAN V03.10  FRI 15-JUL-79 00:08:37
NEWLIB                  FRI 15-JUL-79 00:08:35

MODULE          GLOBALS         GLOBALS         GLOBALS

                COS             SIN
                DATAN           DATAN2
                ATAN            ATAN2
                DCOS            DSIN
```

The command executes in the following sequence:

1. Removes global SQRT from NEWLIB
2. Replaces any duplicates of the modules in the file LIB2.OBJ
3. Inserts the modules in the file LIB3.OBJ
4. Lists the directory of NEWLIB.OBJ on the terminal

LINK

The LINK command converts object modules into a format suitable for loading and execution.



The RT–11 system lets you separately assemble a main program and each of its subroutines without assigning an absolute load address at assembly time. The linker can then process the object modules of the main program and subroutines to relocate each object module and assign absolute addresses. It links the modules by correlating global symbols that are defined in one module and referenced in another, and it creates the initial control block for the linked program. The linker can also create an overlay structure (if you specify the /PROMPT option) and include the necessary run-time overlay handlers and tables. The linker searches libraries you specify to locate unresolved global symbols, and it automatically searches the default system subroutine library, SYSLIB.OBJ, to locate any remaining unresolved globals. Finally, the linker produces a load map (if you specify /MAP) that shows the layout of the executable module. See Chapter 11 for a more detailed explanation of the RT–11 linker. The linker also can produce and STB file.

In the command syntax illustrated above, *filespecs* represents the object modules to be linked. Each input module should be stored on a random-access device (disk, diskette, DECtape, or DECtape II); the output device for the load map file can be any RT–11 device. The output for an .LDA file (if you specify /LDA) can also be any RT–11 device, even those that are not block replaceable, such as paper tape.

The default file types are as follows:

Load Module : .SAV,.REL(/FOREGROUND), .LDA(/LDA)
Map Output : .MAP
Object Module : .OBJ

If you specify two or more files to be linked, separate the files by commas. The system creates an executable file with the same name as the first file in the input list (unless you use /EXECUTE to change it).

The LINK command options and explanations of how to use them follow. Table 4–9 summarizes LINK prompting sequence for commands that combine two or more LINK options.

Table 4–9: Prompting Sequence for LINK Options

Option	Prompt
/TRANSFER	Transfer symbol?
/STACK	Stack symbol?
/EXTEND:n	Extend section?
/BOUNDARY:value	Boundary section?
/ROUND:n	Round section?
/INCLUDE	Library search?

If you combine any of the options listed in Table 4–9, the system prompts you for information in the sequence shown in the table. Note that the *Library search?* prompt is always last. This is the only prompt that accepts more than one line as a response. For all the prompts, terminate your response with a carriage return. Terminate your list of responses to the *Library search?* prompt by typing an extra carriage return. Note that if the command lines are in an indirect file and the system encounters an end-of-file before all the prompting information has been supplied, it prints the prompt messages on the terminal.

/ALLOCATE:size Use this option with /EXECUTE or /MAP to reserve space on the device for the output file. The argument *size* represents the number of blocks of space to allocate. The meaningful range for this value is from 1 to 32767. A value of –1 is a special case that creates the largest file possible on the device. When used with /EXECUTE, /ALLOCATE is valid only when you are generating a .REL or .LDA file.

/ALPHABETIZE When you use this option, the linker lists in the load map your program's global symbols in alphabetical order.

/BITMAP Use this option if you want the linker to create a memory usage bitmap. This is the default setting.

/NOBITMAP Use this option if you do not want the linker to create a memory usage bitmap. This option is useful if you are preparing your program for ROM storage and its code lies between locations 360 and 377 inclusive. /BITMAP is the default setting.

/BOTTOM:n Use this option to specify the lowest address to be used by the relocatable code in the load module. The argument *n* represents a six-digit unsigned, even octal number. If you do not use this option, the linker

positions the load module so that the lowest address is location 1000 (octal). This option is invalid for foreground links.

/BOUNDARY:value Use the /BOUNDARY option to start a specific program section on a particular address boundary. The system generates a whole number multiple of the value you specify for the starting address of the program section. The argument *value* must be a power of 2. The system extends the size of the previous program section to accommodate the new starting address for the specific section. When you have entered the complete LINK command, the system prompts you for the name of the section whose starting address you need to modify. The prompt is:

```
Boundary section?
```

Respond with the appropriate program section name and terminate your response with a carriage return.

/DEBUG[:filespec] Use this option to link ODT (on-line debugging technique, described in Chapter 21) with your program to help you debug it. If you supply the name of another debugging program, the system links the debugger you specify with your program. The system links the debugger low in memory relative to your program.

/EXECUTE[:filespec] Use this option to specify a file name or device for the executable file. Because the LINK command creates executable files by default, the following two commands have the same meaning:

```
.LINK MYPROG

.LINK/EXECUTE MYPROG
```

Both commands link MYPROG.OBJ and produce MYPROG.SAV as a result. The /EXECUTE option has different meanings when it follows the command and when it follows the file specification. The following command creates an executable file called PROG1.SAV on device RK1:.

```
.LINK/EXECUTE:RK1:  PROG1,PROG2
```

The next command creates an executable file called MYPROG.SAV on device DK:.

```
LINK RTN1,RTN2,MYPROG/EXECUTE
```

/NOEXECUTE Use this option to suppress creation of an executable file.

/EXTEND:n This option allows you to extend a program section to a specific octal value *n*. The resultant program section size is equal to or greater than the value you specify, depending on the space the object code requires. When you have entered the complete LINK command, the system prompts you for the name of the program section you need to extend. The prompt is:

```
Extend section?
```

Respond with the appropriate program section name, and terminate your response with a carriage return.

/FILL:n Use this option to initialize unused locations in the load module and place a specific octal value *n* in those locations. Note that the linker automatically initializes to 0 unused locations in the load module; use this option to place another value in those locations. This option can be useful in eliminating random results that occur when a program references uninitialized memory by mistake. It can also help you to determine which locations have been modified by the program and which remain unchanged.

/FOREGROUND[:stacksize] This option produces an executable file in relocatable (.REL) format for use as a foreground job under the FB or XM monitor. You cannot use .REL files under the single-job system. This option assigns the default file type .REL to the executable file. The argument *stacksize* represents the number of bytes of stack space to allocate for the foreground job. The value you supply is interpreted as an octal number; specify an even number. Follow *n* with a decimal point (*n.*) to represent a decimal number. The default value is 128 (decimal) (or 200 octal) bytes of stack space. DIGITAL recommends that you allocate 256. bytes of stack space when linking a FORTRAN program to run in the foreground.

/INCLUDE This option lets you take global symbols from any library and include them in the linked memory image. When you have entered the complete LINK command, the system prompts you for a list of global symbols to include in the load module. The prompt is:

```
Library search?
```

Respond by typing the global symbols to be included in the load module. Type a carriage return after each global symbol. Type a carriage return on a line by itself to terminate the list. This option forces modules that are not called by other modules to be loaded from the library.

/LDA This option produces an executable file in LDA format. The LDA-format file can be output to any device, including those that are not block-replaceable, such as the paper tape punch or cassette. The default file type .LDA is assigned by /LDA to the executable file. This option is useful for files that you need to load with the Absolute Binary Loader.

/LIBRARY This option is the same as /LINKLIBRARY. It is included here only for system compatibility.

/LINKLIBRARY:filespec You can use this option to include the library file you specify as an object module library in the linking operation. Because the system automatically recognizes library files in the linking operation you do not normally need this option; it is provided for compatibility with the EXECUTE command.

/MAP[:filespec] You must specify this option to produce a load map listing. The /MAP option has different meanings depending on where you put it in the command line.

If you specify /MAP without a filespec in the list of options that immediately follows the command name, the system generates a listing that prints on the line printer. If you follow /MAP with a device name, the system creates a

map file on that device. If the device is a file-structured device, the system stores the listing file on that device, assigning it the same name as the first input file and a .MAP file type. The following command produces a load map on the terminal.

```
.LINK/MAP:TT:  MYPROG
```

The next command creates a map listing file called MYPROG.MAP on RK3:.

```
.LINK/MAP:RK3:  MYPROG
```

If the /MAP option contains a name and file type to override the default of .MAP, the system generates a listing with that name. The following command, for example, links PROG1 and PROG2, producing a map listing file called MAP.OUT on device DK:.

```
.LINK/MAP:MAP.OUT PROG1,PROG2
```

Another way to specify /MAP is to type it after the file specification to which it applies. To link a file and produce a map listing file with the same name, use a command similar to this one.

```
.LINK PROG1,PROG2/EXECUTE/MAP
```

The command shown above links PROG1 and PROG2, producing files PROG2.SAV and PROG2.MAP. If you specify a file name on a /MAP option following a file specification in the command line, it has the same meaning as when it follows the command.

/PROMPT Use this option to enter additional lines of input. The system continues to accept lines of linker input until you enter two slashes (//). Chapter 11 describes the commands you can enter directly to the linker. When you use the /PROMPT option, note that successive lines of input must conform to CSI conventions (see Chapter 6, Command String Interpreter). The example that follows uses the /PROMPT option to create an overlay structure for the program COSINE.MAC:

```
.LINK/PROMPT COSINE
*TAN/O:1
*COS1/O:1
*SIN3/O:2
*LML3/O:2//
```

The /PROMPT option also gives you a convenient way to create an overlaid program from an indirect file. The file ANTON.COM contains these lines:

```
A/PROMPT
SUB1/O:1
SUB2/O:1
SUB3,SUB4/O:1
//
```

The following command produces an executable file, DK:A.SAV, and a link map on the printer.

```
.LINK/MAP  @ANTON
```

/ROUND:n This option rounds up the section you specify so that the size of the root segment is a whole number of the value *n* you supply. The argument *n* must be a power of 2. When you have entered the complete LINK command, the system prompts you for the name of the section that you need to round. The prompt is:

```
Round section?
```

Respond with the appropriate program section name, and terminate your response with a carriage return.

/RUN Use this option to initiate execution of the resultant .SAV file. This option is valid for background jobs only. Do not use /RUN with any option that requires a response from the terminal.

/SLOWLY This option instructs the system to allow the largest possible memory area for the link symbol table at the expense of making the link process slower. Use this option only if an attempt to link a program failed because of symbol table overflow.

/STACK[:n] This option lets you modify the stack address, location 42, which is the address that contains the value for the stack pointer. When your program executes, the monitor sets the stack pointer (SP) to the contents of location 42. The argument *n* is an even, unsigned, six-digit, octal number that defines the stack address. When you have entered the complete LINK command, the system prints the following prompt message if you did not already specify a value for *n*:

```
Stack symbol?
```

Respond with the global symbol whose value is the stack address. You cannot specify a number at this point. Terminate your response with a carriage return. If you specify a nonexistent symbol, the system prints an error message. It then sets the stack address to 1000 (for memory image files) or to the bottom address if you used /BOTTOM.

/SYMBOLTABLE[:filespec] When you use this option, the linker creates a file that contains symbol definitions for all the global symbols in the load module. Enter the symbol table file specification as the third output specification in the LINK command line. If you do not specify a file name, the linker uses the name of the first input file and assigns a .STB file type. By default, the system does not create a symbol table file.

The following example creates the symbol table file BTAN.STB.

```
.LINK   BOBJ,BOBJ2      AOBJ,BOBJ/SYMBOLTABLE:BTAN
```

/TOP:value Use this option to specify the highest address to be used by the relocatable code in the load module. The argument *value* represents an unsigned, even octal number.

/TRANSFER[:n] The transfer address is the address at which a program starts when you initiate execution with R, RUN, FRUN, or SRUN. The /TRANSFER option lets you specify the start address of the load module.

The argument *n* is an even, unsigned, six-digit, octal number that defines the transfer address. When you have entered the complete LINK command, the system prints the following prompt message if you did not already specify a value for *n*:

```
Transfer symbol?
```

Respond with the global symbol whose value is the transfer address. You cannot specify a number at this point. Terminate your response with a carriage return. If you specify a nonexistent symbol, an error message prints and the linker sets the transfer address to 1 so that the system cannot execute the program. If the transfer address you specify is odd, the program does not execute after loading, and control returns to the monitor.

/WIDE Use this option with /MAP to produce a wide load map listing. Normally, the listing is wide enough for three Global Value columns, which is suitable for paper with 72 or 80 columns. The /WIDE option produces a listing that is six Global Value columns wide, which is equivalent to 132 columns.

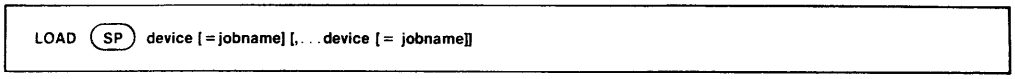
Table 4–9 lists the sequence in which the system prompts you for additional information when you combine LINK options.

/XM When you use this option, you enable special .SETTOP and .LIMIT features provided in the XM monitor. This option allows a virtual job to map a scratch region in extended memory with the .SETTOP programmed request. See the *RT–11 Programmer's Reference Manual* and the *RT–11 Software Support Manual*, for more details on these special features. Do not use with /FOREGROUND.

If you want to create an extended memory overlay structure for your program, use the /PROMPT option. You can then specify on subsequent lines the overlay structure using the LINK /V option (see Chapter 11 of this manual). Note that when you use /V to create an overlay structure, the linker automatically enables the special .SETTOP and .LIMIT features.

LOAD

The LOAD command loads a device handler into memory for use with foreground, background, or system jobs, or BATCH.



In the command syntax shown above, *device* represents the device handler to be made resident; *jobtype* assigns the device handler to the background job if it has the value B, or to the foreground or system job if it has the value F. The *jobtype* specification is invalid with the SJ monitor. Under a monitor that has system job support, *jobtype* can be the logical job name of a system job.

The LOAD command helps control system execution by bringing a device handler into memory and optionally allocating the device to a job. The system allocates memory for the handler as needed. Before you use a device in a foreground program with the FB monitor, or any device at all with the XM monitor, you must first load the device handler. A device can be owned exclusively by either the foreground, background, or system job. (Note that BATCH, if running, is considered to be a background job under the FB and XM monitors.) This exclusive ownership prevents the input and output of two different jobs from being intermixed on the same non file-structured device. In the following example, magtape belongs to the background job, while DECtape is available for use by either the background, foreground, or system job; the line printer is owned by the foreground job. All three handlers are made resident in memory.

```
.LOAD DT:,MT:=B,LP:=F
```

For a monitor with system job support, the following example reserves the line printer for the system job QUEUE.

```
.LOAD LP:=QUEUE
```

Different units of the same random-access device controller can be owned by different jobs. Thus, for example, DT1: can belong to the background job, while DT5: can belong to the foreground or system job. If no ownership is indicated, the device is available for public use.

NOTE

If you use the LOAD command to load a non-file-structured device handler, and assign ownership of that handler to a give example job, all units of that particular device become assigned to that job. This means, no other job can use any unit of that particular device.

To change ownership of a device, use another LOAD command. It is not necessary to first unload the device. For example, if the line printer has been loaded into memory and assigned to the foreground job as in the example

above, the following command reassigns it to the background job without unloading the handler first.

```
.LOAD LP:=B
```

Note, however, that if you interrupt an operation that involves magtape or cassette, you must unload (with the UNLOAD command) then load the appropriate device handler (MM, MT, MS, or CT). When using the MT handler with the FB monitor, this restriction does not apply.

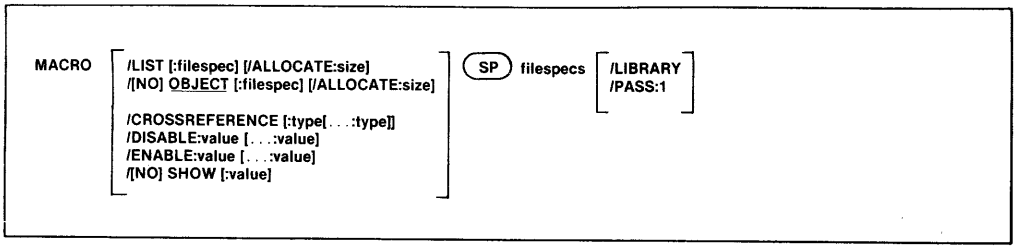
You cannot assign ownership of the system unit (the unit you bootstrapped) of a system device, and any attempt to do so is ignored. You can, however, assign ownership of other units of the same type as the system device. LOAD is valid for use with logical names. For example:

```
.ASSIGN RK: XY
.LOAD XY:=F
```

If you are using a diskette, loading the necessary device handlers into memory can improve system performance significantly, since no handlers need to be loaded dynamically from the diskette. Use the SHOW command to display on the terminal the status of device handlers and device ownership.

MACRO

The MACRO command invokes the MACRO assembler to assemble one or more source files.



In the command syntax shown above, *filespecs* represents one or more files to be included in the assembly. If you omit a file type for an input file, the system assumes .MAC. Output default file types are .LST for listing files and .OBJ for object files.

To assemble multiple source files into a single object file, separate the files by plus (+) signs in the command line. Unless you specify otherwise, the system creates an object file with the same name as the first input file and gives it an .OBJ file type. To assemble multiple files in independent assemblies, separate the files by commas (,) in the command line. This generates a corresponding object file for each set of input files.

Language options are position-dependent — that is, they have different meanings depending on where you place them in the command line. Options that qualify a command name apply across the entire command string. Options that follow a file specification apply only to the file (or group of files separated by plus signs) that they follow in the command string.

You can enter the MACRO command as one line, or you can rely on the system to prompt you for information. The MACRO command prompt is *Files?* for the input specification. The system prints on the terminal the number of errors MACRO detects during an assembly, as this printout shows:

```
.MACRO/CROSSREFERENCE PROG1+PROG2/LIST/OBJECT
ERRORS DETECTED:  0
```

Chapter 10 and the *PDP–11 MACRO Language Reference Manual* contain more detailed information about using MACRO. The options you can use with the MACRO command follow.

/ALLOCATE:size Use this option with /LIST or /OBJECT to reserve space on the device for the output file. The argument *size* represents the number of blocks of space to allocate. The meaningful range for this value is from 1 to 32767. A value of –1 is a special case that creates the largest file possible on the device.

/CROSSREFERENCE[:type[...:type] Use this option to generate a symbol cross-reference section in the listing. This information is useful for program maintenance and debugging. Note that the system does not generate a

listing by default. You must also specify /LIST in the command line to get a cross-reference listing. The argument *type* represents a one-character code that indicates which sections of the cross-reference listing the assembler should include. Table 4–10 summarizes the arguments and their meaning.

Table 4–10: Cross-reference Sections

Argument	Section Type
S	User-defined symbols
R	Register symbols
M	Macro symbolic names
P	Permanent symbols (instructions, directives)
C	Control sections (.CSECT symbolic names)
E	Error codes
no argument	Equivalent to :S:M:E

/DISABLE:value[...:value] Use this option to specify a MACRO .DSABL directive. See the *PDP–11 MACRO Language Reference Manual* for a description of the directive and a list of all legal values. Table 4–11 summarizes the arguments and their meaning.

Table 4–11: .DSABL and .ENABL Directive Summary

Argument	Default	Enables or Disables
ABS	disable	Absolute binary output
AMA	disable	Assembles all absolute addresses as relative addresses
CDR	disable	Treats source columns 73 and greater as comments
FPT	disable	Floating-point truncation
GBL	disable	Treats undefined symbols as globals
LC	disable	Accepts lower case ASCII input
LSB	disable	Local symbol block
PNC	enable	Binary output
REG	enable	Mnemonic definitions of registers

/ENABLE:value[...:value] Use this option to specify a MACRO .ENABL directive. See the *PDP–11 MACRO Language Reference Manual* for a description of the directive and a list of all legal values. Table 4–11 summarizes the arguments and their meaning.

/LIBRARY This option identifies the file it qualifies as a library file; use it only after a library file specification in the command line. The MACRO assembler looks first to the library file or files you specify and then to the system library, SYSMAC.SML, to satisfy references (made with the .MCALL directive) from MACRO programs. In the example below, the command string includes two user libraries.

```
.MACRO MYLIB1/LIBRARY+A+MYLIB2/LIBRARY+B
```

When MACRO assembles file A, it looks first to the library, MYLIB1.MAC, and then to SYSMAC.SML to satisfy .MCALL references. When it assem-

bles file B, MACRO searches MYLIB2.MAC, MYLIB1.MAC, and then SYSMAC.SML, in that order, to satisfy references.

/LIST[:filespec] You must specify this option to produce a MACRO assembly listing. The /LIST option has different meanings depending on where you place it in the command line.

The /LIST option produces a listing on the line printer when /LIST follows the command name. For example, the following command line produces a line printer listing after compiling a MACRO source file:

```
.MACRO/LIST MYPROG<RET>
```

When the /LIST option follows the file specification, it produces a listing file. For example, the following command line produces the listing file DK:MYPROG.LST after compiling a MACRO source file:

```
MACRO MYPROG/LIST<RET>
```

If you specify /LIST without a file specification in the list of options that immediately follows the command name, the MACRO assembler generates a listing that prints on the line printer. If you follow /LIST with a device name, the system creates a listing file on that device. If the device is a file-structured device, the system stores the listing file on that device, assigning it the same name as the input file and a .LST file type. The following command produces a listing on the terminal.

```
.MACRO/LIST:TT: A
```

The next command creates a listing file called A.LST on RK3:.

```
.MACRO/LIST:RK3: A
```

If the /LIST option contains a name and file type to override the default of .LST, the system generates a listing file with that name. The following command for example, assembles A.MAC and B.MAC together, producing files A.OBJ and FILE1.OUT on device DK:.

```
.MACRO/LIST:FILE1.OUT A+B
```

You cannot use a command like the next one. In this example, the second listing file would replace the first one and cause an error.

```
.MACRO/LIST:FILE2 A,B
```

Another way to specify /LIST is to type it after the file specification to which it applies. To produce a listing file with the same name as a particular input file, you can use a command similar to this one:

```
.MACRO A+B/LIST:RK3:
```

The above command assembles A.MAC and B.MAC, producing files DK:A.OBJ and RK3:B.LST. If you specify a file name on a /LIST option following a file specification in the command line, it has the same meaning as

when it follows the command. The following two commands have the same results:

```
.MACRO A/LIST:B

.MACRO/LIST:B A
```

Both commands generate output files A.OBJ and B.LST.

Remember that file options apply only to the file (or group of files that are separated by plus signs) they follow in the command string. For example:

```
.MACRO A/LIST,B
```

This command assembles A.MAC, producing A.OBJ and A.LST. It also assembles B.MAC, producing B.OBJ. However, it does not produce any listing file for the assembly of B.MAC.

/OBJECT[:filespec] Use this option to specify a file name or device for the object file. Because MACRO creates object files by default, the following two commands have the same meaning:

```
.MACRO A

.MACRO/OBJECT A
```

Both commands assemble A.MAC and produce A.OBJ as output. The /OBJECT option functions like the /LIST option; it can be either a command or a file qualifier.

As a command option, /OBJECT applies across the entire command string. The following command, for example, assembles A.MAC and B.MAC separately, creating object files A.OBJ and B.OBJ on RK1:.

```
.MACRO/OBJECT:RK1: A,B
```

Use /OBJECT as a file option to create an object file with a specific name or destination. The following command assembles A.MAC and B.MAC together, creating files B.LST and B.OBJ.

```
.MACRO A+B/LIST/OBJECT
```

/NOOBJECT Use this option to suppress creation of an object file. As a command option, /NOOBJECT suppresses all object files; as a file option, it suppresses only the object file produced by the related input files. In this command, for example, the system assembles A.MAC and B.MAC together, producing files A.OBJ and B.LST. It also assembles C.MAC and produces C.LST, but does not produce C.OBJ.

```
.MACRO A+B/LIST,C/NOOBJECT/LIST
```

/PASS:1 Use this option on a prefix macro file to process that file during pass 1 of the assembly only. This option is useful when you assemble a source program together with a prefix file that contains only macro definitions, since these definitions do not need to be redefined in pass 2 of the

assembly. The following command assembles a prefix file and a source file together, producing files PROG1.OBJ and PROG1.LST.

```
.MACRO PREFIX.MAC/PASS:1+PROG1/LIST/OBJECT
```

/SHOW:value Use this option to specify any MACRO .LIST directive. The *PDP–11 MACRO Language Reference Manual* explains how to use these directives. Table 4–12 summarizes the arguments and their meaning. Note that you must explicitly request a listing file with the /LIST option.

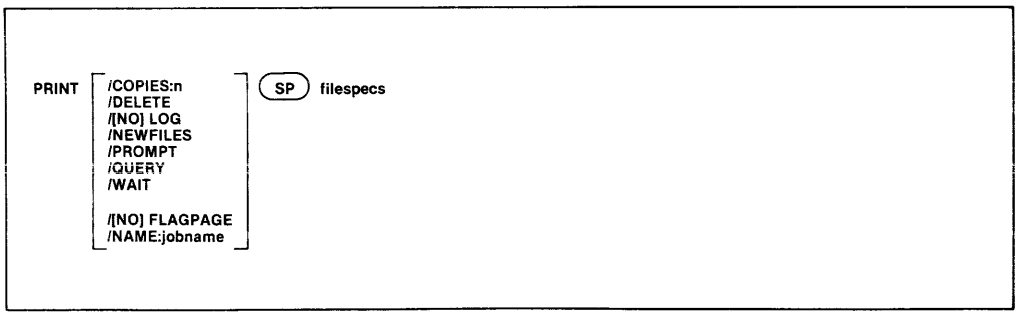
Table 4–12: .LIST and .NLIST Directive Summary

Argument	Default	Controls
SEQ	list	Source line sequence numbers
LOC	list	Location counter
BIN	list	Generated binary code
BEX	list	Binary extensions
SRC	list	Source code
COM	list	Comments
MD	list	Macro definitions, repeat range expansions
MC	list	Macro calls, repeat range expansions
ME	nolist	Macro expansions
MEB	nolist	Macro expansions binary code
CND	list	Unsatisfied conditionals, .IF and .ENDC statements
LD	nolist	Listing directives with no arguments
TOC	list	Table of Contents
TTM	line printer mode	Output format
SYM	list	Symbol table

/NOSHOW:value Use this option to specify any MACRO .NLIST directive. The *PDP–11 MACRO Language Reference Manual* explains how to use these directives. Table 4–12 summarizes the valid arguments and their meaning. Note that you must explicitly request a listing file with the /LIST option.

PRINT

The PRINT command lists the contents of one or more files on the line printer.



In the command syntax illustrated above, *filespecs* represents the file or files to be printed. You can explicitly specify up to six files as input to the PRINT command. The system prints the files in the order in which you specify them in the command line. You can also use wildcards in the file specification. In this case, the system prints the files in the same order that they occur in the directory of the specified device. If you specify more than one file, separate the files by commas. If you omit the file type for a file specification, the system assumes .LST. You can specify the entire command on one line, or you can rely on the system to prompt you for information. The PRINT command prompt is *Files?*.

If you are running QUEUE as either a foreground or system job, many of the PRINT commands are executed by this program, therefore, the keyboard monitor may return the dot prompt immediately. See Chapter 20, Queue Package, for more information. If QUEUE is not running, some PRINT options are invalid (as noted). Likewise, some PRINT options are invalid if QUEUE is running. You should use the LOAD command to assign ownership of a non-file structured device to QUEUE so that another job and QUEUE will not intermix output on that device.

The PRINT command options follow; they include command examples.

/COPIES:n Use this option to print more than one copy of the file. The meaningful range of values for the decimal argument *n* is from 1 to 32 (1 is the default). This option must appear immediately after the PRINT command, and not after the file specification. The following command, for example, prints three copies of the file REPORT.LST on the line printer.

```
.PRINT/COPIES:3 REPORT
```

/DELETE Use this option to delete a file after it lists on the line printer. This option must appear following the command in the command line. The PRINT/DELETE operation does not ask you for confirmation before it executes. You must use /QUERY for this function. The following example prints PROG1.BAS on the line printer, then deletes it from DX1:.

```
.PRINT/DELETE DX1:PROG1.BAS
```

/FLAGPAGE:n Use this option if you want banner pages for each file being printed, where *n* represents the number of banner pages you want for each file. This option is valid only if you are running QUEUE. If you specify more than one file to be printed, QUEUE prints a banner page for each file.

The banner page that QUEUE creates consists of a page showing the file name in large, block letters. The banner page also includes a trailer that lists the job name, the date and time the job was output, the copy number and number of copies in the job, and the input file specification.

NOTE

If you use the PRINT command to output files, and QUEUE is running, you may get banner pages even when you do not specify /FLAGPAGE. This condition is due to a default value you can set when you run QUEMAN, the background job that serves as an interface between you and QUEUE. The QUEMAN /P option sets the default number of banner pages for output jobs, so that each time you output a job, you get banner pages. This condition remains in effect until you reset it with the QUEMAN /P option. For more information on QUEMAN and the /P option, see Chapter 20, Queue Package.

The following example prints three banner pages for each file in the command line.

```
.PRINT/FLAGPAGE:3 PROG1.MAC,PROG1.LST,PROG1.STB
```

/NOFLAGPAGE Use this option if you do not want any banner pages printed for each of the files in the job you want printed. Use this option only if you are running QUEUE. This option is useful if you have previously set QUEMAN's /P option to create banner pages each time a job is output (see note above). The default setting is /NOFLAGPAGE unless you specify otherwise with the QUEMAN /P option.

/LOG This option lists on the terminal the names of the files that are printed by the current command. Normally, the system prints a log only if there is a wildcard in the file specification. If you specify /QUERY, the query messages replace the log, unless you specifically type /LOG/QUERY in the command line. The following example shows a PRINT command and the resulting log.

```
.PRINT/LOG/DELETE REPORT
 Files copied/deleted:
DK:REPORT.LST to LP:
```

This option is invalid if QUEUE is running.

/NOLOG This option prevents a list of the files copied from typing out on the terminal. You can use this option to suppress the log when you use a wildcard in the file specification. This option is invalid if QUEUE is running.

/NAME:[dev:]jobname Use this option to specify a job name for the files you want printed. This option is valid only if you are running QUEUE. You can use up to six alphanumeric characters for the job name. If you do not use the /NAME option, the system uses the first input file name as the job name. If you specify a device with the job name, you can send the files to that device, permitting you to send files to any valid RT–11 device. If you send the files to a mass storage volume, the system uses the job name as the file name for the job, assigning a .JOB file type. Note that the handler for the output device must be loaded in memory (see the LOAD command description).

The following example sends JOB5, consisting of FILE1.LST, FILE2.LST, and FILE3.LST, to DX1:.

```
.PRINT/NAME:DX1:JOB5 FILE1,FILE2,FILE3
```

The files from this example reside on DX1: as JOB5.JOB.

/NEWFILES Use this option in the command line if you need to print only those files that have the current date. The following example shows a convenient way to print all new files after a session at the computer.

```
.PRINT/NEWFILES *.LST
 Files copied:
DK:OUTFIL.LST    to LP:
DK:REPORT.LST    to LP:
```

This option is invalid if QUEUE is running.

/PROMPT Use this option to continue a command string onto subsequent lines. This option is valid only if you are running QUEUE. When you use /PROMPT, you can enter file specifications on subsequent lines directly to QUEMAN, described in Chapter 20. Terminate the command with two slashes (//).

The following example uses /PROMPT to print FILE1,FILE2,FILE3,FILE4, and FILE5:

```
.PRINT/PROMPT FILE1
*FILE2, FILE3
*FILE4
*FILE5//
```

/QUERY If you use this option, the system requests confirmation from you before it performs the operation. /QUERY is particularly useful on operations that involve wildcards, when you may not be sure which files the system selected for an operation. Note that if you specify /QUERY in a PRINT command line that also contains a wildcard in the file specification, the confirmation messages that print on the terminal replace the log messages that would normally appear. You must respond to a query message by typing Y (or anything that begins with Y) and a carriage return to initiate execution of a particular operation. The system interprets any other

response to mean NO; it does not perform the specific operation. The following example uses /QUERY.

```
.PRINT/QUERY *.LST
 Files copied:
DK:OUTFIL.LST    to LP:? N
DK:REPORT.LST    to LP:? Y
```

This option is invalid if QUEUE is running.

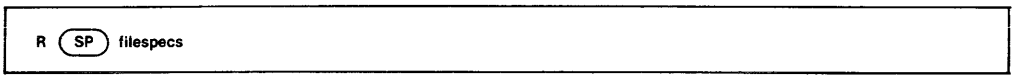
/WAIT This option is useful if you have a single-disk system. When you use this option, the system initiates the PRINT operation, but then pauses and waits for you to mount the volume from which you want the operation to take place. When the system pauses, it prints *Mount input volume in <device>; Continue?*. When the volume is mounted, type Y followed by a carriage return.

The following command line prints ERREX.MAC from RK0:

```
.PRINT/WAIT RK0:ERREX.MAC
Mount input volume in RK0:; Continue?Y
Mount system volume in RK0:; Continue?Y
```

In the case of PRINT, the system prints the file or files you specify before it prints *Mount system volume in <device>; Continue?*. Make sure when you use /WAIT that PIP is on the system volume. This option is invalid if QUEUE is running.

The R command loads a memory image file from the system device into memory and starts execution.



In the command syntax shown above, *filespec* represents the program to be executed. The default file type is .SAV. The only valid device is SY:. The R command is similar to the RUN command except that the file you specify in an R command string must be on the system device (SY:). Use the R command only with background jobs including privileged jobs in XM. (Use FRUN to execute a foreground job under the FB or XM monitor.) The following command loads and executes MYPROG.SAV from device SY:.

```
.R MYPROG
```

The R command is the only monitor command that can execute a background virtual job under the XM monitor. The R command creates a virtual memory partition for the job, creates a region 0 and window 0 definition block, and sets up the user mapping registers.

REENTER

The REENTER command starts the program at its reentry address (the start address minus 2).

```
REENTER
```

The REENTER command accepts no options or arguments. REENTER does not clear or reset any memory areas. Use it to avoid reloading the same program for subsequent execution. You can use REENTER to return to a system program or to any program that allows for a REENTER after the program terminates. You can also use REENTER after you have used two CTRL/Cs to interrupt those programs.

If you issue the REENTER command and it is not valid, the message *?KMON-F-Illegal command* is printed. You must start that program with an R or RUN command.

In the following example the directory program (DIR) lists the directory of DK: on the line printer. Two CTRL/Cs interrupt the listing and return to the monitor. REENTER starts DIR at its reentry address, and DIR prompts for a line of input.

```
.R DIR
*LP:=DK:*.*
^C
^
.
.REENTER
*
```

Note in the example above that using REENTER does not mean that the directory listing continues from where it was interrupted, only that the DIRECTORY program re-commences execution.

REMOVE

The REMOVE command removes a device name from the system tables.

```
REMOVE (SP) device [,...device]
```

In the command syntax shown above, *device* represents the device to be removed from the system tables. The REMOVE command accepts no options. You can enter the REMOVE command on one line, or you can rely on the system to prompt you for information. The REMOVE command prompt is *Device?*.

Using the REMOVE command does not change the monitor disk image; it only modifies the system tables of the monitor currently in core. This allows you to configure a special system for a single session at the computer without having to reconfigure to return to your standard device configuration. Bootstrapping the system device restores the original device configuration. To permanently REMOVE a device, include the REMOVE command in the standard system startup indirect command file.

You cannot remove SY: (the handler for the system device), BA: (the BATCH handler), or TT: (the terminal handler). If you attempt to REMOVE a device that does not exist in the running monitor's system table, the system prints an error message. You can use the INSTALL command to install a new device after using the REMOVE command to remove a device (thus creating a free device slot).

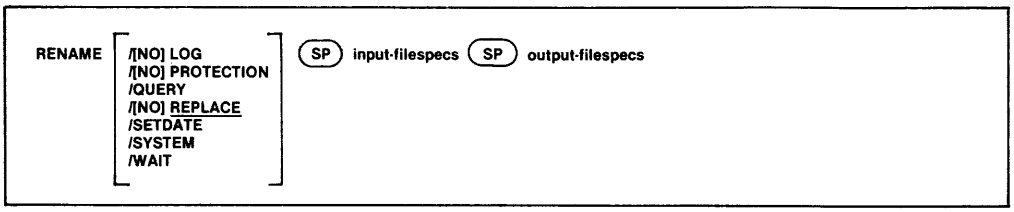
The following command removes the line printer handler and the card reader handler from the system. Note that the colons (:) are optional.

```
.REMOVE  LP:,CR:
```

Use the SHOW command to display on the terminal a list of devices that are currently available on your system.

RENAME

The RENAME command assigns a new name to an existing file.



In the command syntax illustrated above, *input-filespecs* represents the files to be renamed, and *output-filespec* represents the new name. You can specify up to six input files, but only one output file. Note that the device specification must be the same for input and output; you cannot rename a file from one device to another. If a file exists with the same name and file type as the output file you specify, the system deletes the existing file unless you use the /NOREPLACE option to prevent this.

So that you do not rename system (.SYS) files by accident when you use a wildcard in the file specification, the system requires you to use the /SYSTEM option when you need to rename system files. To rename files that cover bad blocks (.BAD files), you must explicitly give the file name and file type of the specified .BAD file. Since .BAD files cover bad blocks on a device, you usually do not need to rename or otherwise manipulate these files.

Note that because of the file protection feature, you cannot execute any RENAME operations that result in deleting a protected file. For example, you cannot rename a file to the name of a protected file that already exists on the same volume.

The options you can use with the RENAME command follow.

/LOG This option lists on the terminal the files that were renamed by the current command. Normally, the system prints a log only if there is a wildcard in the file specification. If you specify /QUERY, the query messages replace the log (unless you specifically type /LOG/QUERY in the command line).

This example demonstrates logging.

```
.RENAME DX0:(A*.MAC  *.FOR)
 Files renamed:
DX0:ABC.MAC     to DX0:ABC.FOR
DX0:AAF.MAC     to DX0:AAF.FOR
```

/NOLOG This option prevents a list of the files that are renamed from appearing on the terminal.

/NEWFILES Use this option in the command line if you want to rename only those files that have the current date. This is a convenient way to access all new files after a session at the computer.

/PROTECTION Use this option to give a file *protected* status so that it cannot be deleted until you disable that status. Note that if a file is protected, you cannot delete it implicitly. For example, you cannot perform any operations on a file that result in deleting a protected file. You can change a protected file's name, but not its protected status, unless you also use the /NOPROTECTION option.

/NOPROTECTION Use this option to enable a file for deletion. This option disables a file's *protected* status.

/QUERY If you use this option, the system requests confirmation from you before it performs the operation. /QUERY is particularly useful on operations that involve wildcards, when you may not be sure which files the system selected for the operation. Using the /QUERY option also provides a quick way of performing operations on several files. For example, renaming several files is easier if you use /QUERY. You can then specify Y for each file you want renamed, as the following example shows.

```
.RENAME/QUERY  *.BAK  *.MAC
 Files renamed:
DK:PROG1.BAK    to DK:PROG1.MAC ? Y
DK:PROG2.BAK    to DK:PROG2.MAC ? Y
DK:PROG6.BAK    to DK:PROG6.MAC ? Y
DK:LML8A.BAK    to DK:LML8A.MAC ?
DK:LML9 .BAK    to DK:LML9 .MAC ? Y
```

Note that if you specify /QUERY in a command line that also contains a wildcard in the file specification, the confirmation messages that print on the terminal replace the log messages that would normally appear. You must respond to a query message by typing Y (or anything that begins with Y) and a carriage return to initiate execution of a particular operation. The system interprets any other response to mean NO; it does not perform the specific operation. The following example demonstrates querying.

```
.RENAME/QUERY DX0:(PIP1.SAV PIP.SAV)
 Files renamed:
DX0:PIP1.SAV    to DX0:PIP.SAV    ? Y
```

/REPLACE This is the default mode of operation for the RENAME command. If a file exists with the same name as the file you specify for output, the system deletes that duplicate file when it performs the rename operation.

/NOREPLACE This option prevents execution of the rename operation if a file with the same name as the output file you specify already exists on the same device. The following example uses /NOREPLACE. In this case, the output file already existed and no action occurs.

```
.RENAME/NOREPLACE DX0:TEST.SAV DX0:DUP.SAV
?PIP-W-Output file found, no operation performed DX0:TEST.SAV
```

/SETDATE This option causes the system to put the current date on all files it renames, unless the current system date is not set. Normally, the sys-

tem preserves the existing file creation date when it renames a file. The following example renames files and changes their dates.

```
.RENAME/SETDATE DX0:(*.FOR *.OLD)
 Files renamed:
DX0:ABC.FOR     to DX0:ABC.OLD
DX0:AAF.FOR     to DX0:AAF.OLD
DX0:MERGE.FOR   to DX0:MERGE.OLD
```

/SYSTEM Use this option if you need to rename system (.SYS) files. If you omit this option, the system files are excluded from the rename operation and a message is printed on the terminal to remind you of this. This example renames MM.SYS to MX.SYS.

```
.RENAME/SYSTEM DX0:MM.SYS DX0:MX.SYS
```

/WAIT This option is useful if you have a single-disk system. When you use this option, the system initiates the RENAME operation, but then pauses and waits for you to mount the input volume on which the operation is to take place. When the system pauses, it prints *Mount input volume in <device>; Continue?* where *<device>* represents the device into which you mount the volume. When the volume is mounted, type Y followed by a carriage return.

The following command line renames PRIAM.TXT to NESTOR.TXT. PRIAM.TXT is on an RK05 disk.

```
.RENAME/WAIT/NOLOG RK0:PRIAM.TXT NESTOR.TXT
Mount input volume in RK0:; Continue?Y
Mount system volume in RK0:; Continue?Y
```

RESET

The RESET command resets several background system tables and does a general clean-up of the background area.

```
RESET
```

The RESET command accepts no options or arguments.

It causes the system to purge all open input/output channels, initialize the user program memory area, and release any device handlers that were not explicitly made resident with the LOAD command. It also disables CTRL/O, clears locations 40–53, and resets the KMON (keyboard monitor) stack pointer. Use RESET before you execute a program if a device or the monitor needs reinitialization, or when you need to discard the results of previously issued GET commands. The RESET command has no effect on the foreground or system job. The following example uses the RESET command before running a program.

```
.RESET
.R MYPROG
```

RESUME

The RESUME command continues execution of the foreground or system job from the point at which a SUSPEND command was issued.

If you have system job support enabled on your monitor, the RESUME command must be followed by the name of the foreground or system job you wish to resume. (The RESUME command accepts logical job names.) If you do not have system job support enabled on your monitor, do not include the name of the foreground job you wish to resume. When you issue the RESUME command, the foreground or system job enters any completion routines that were scheduled while the job was suspended. Note that RESUME is valid only with the FB and XM monitors. The following command resumes execution of the foreground job that is currently suspended.

```
.RESUME
```

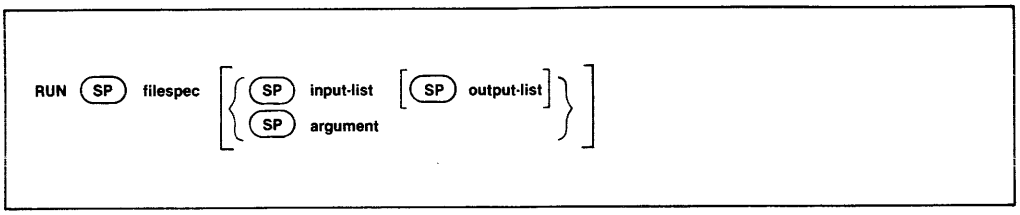
The next command resumes execution of the system job, QUEUE.SYS, that is currently suspended.

```
.RESUME QUEUE
```

You can also use the RESUME command to start a foreground job that you loaded with FRUN using /PAUSE. Likewise, you can use RESUME to start a system job that you loaded with SRUN using /PAUSE.

RUN

The RUN command loads a memory image file into memory and starts execution.



In the command syntax illustrated above, *filespec* represents the program to be executed. The system assumes a .SAV file type for the executable file, which can reside on any RT–11 block-replaceable device. The default device is DK:. The RUN command automatically loads the device handler for the device you specify if it is not already resident. This eliminates the need to explicitly load a device handler when you run an overlaid program from a device other than the system device. The RUN command executes only those programs that have been linked to run as background jobs. (Use FRUN to execute foreground jobs under the FB or XM monitor.)

RUN is a combination of the GET and START commands. First it loads a memory image file from a storage device into memory. Then it begins execution at the program's transfer address. You can use RUN to execute a privileged job under the XM monitor the same way you execute any other background job in FB or SJ. However, a virtual job in XM requires special preparation for execution. You must use the R command to execute a background virtual job. The R command creates a virtual memory partition for the job, creates a region 0 and window 0 definition block for the partition, and sets up the user mapping registers. The following command, for example, executes MYPROG.SAV, which is stored on device DX1:.

```
.RUN DX1:MYPROG
```

You can also pass an argument in the RUN command to the program, or specify a list of input and output. This allows you to specify a line of input for a user program or for a system utility program (which accepts file specifications in the special syntax described in Chapter 6). The system automatically converts the input list and the output list you specify into a format that the Command String Interpreter accepts. For example, to execute the directory program (DIR) and obtain a complete listing of the directory of DX1: on the printer, you can use the following command.

```
.RUN DIR DX1:*.* LP:/E
.
```

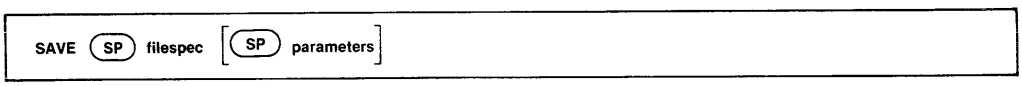
This command has the same effect as the following lines.

```
.RUN DIR
*LP:/E=DX1:*.*
*^C
.
```

Note that when you use either an argument or an input list and output list with RUN, control returns to the monitor when the program completes.

SAVE

The SAVE command writes memory areas in memory image format to the file and device that you specify.



In the command syntax shown above, *filespec* represents the file to be saved on a block-replaceable device. If you do not specify a file type, the system uses .SAV. The parameters represent memory locations to be saved.

Parameters are of the form:

address[-address(2)][,address(3)[-address(n)]]

where:

address is an octal value representing a specific block of memory locations to be saved. If you specify more than one address, each address must be higher than the previous one

RT–11 transfers memory in 256-word blocks, beginning on boundaries that are multiples of 256 (decimal). If the locations you specify make a block that is less than 256 words, the system saves additional words to make a 256-word block

The system saves memory from location 0 to the highest memory address specified by the parameter list or to the program high limit (location 50 in the system communication area). Initially, the system gives the start address and the Job Status Word the default value 0 and sets the stack to 1000. If you want to change these or any of the following addresses, you can use the Deposit command to alter them and the SAVE command to save the correct areas.

Area	Location
Start address	40
Stack	42
JSW	44
USR address	46
High address	50
Fill characters	56

If you change the values of the addresses, it is your responsibility to reset them to their default values. For more information concerning these addresses refer to the *RT–11 Programmer's Reference Manual.* Note that the SAVE command does not write the overlay segments of programs; it saves only the root segment. You cannot use the SAVE command for foreground or virtual jobs.

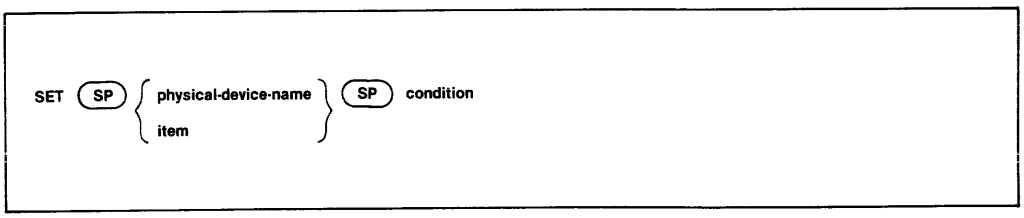
The following command saves locations 10000 through 11777, and 14000 through 14777. It stores the contents of these locations in the file FILE1.SAV on device DK:.

```
.SAVE FILE1 10000-11000,14000-14100
```

The next example sets the reenter bit in the JSW and saves locations 1000 through 5777 in file PRAM.SAV on device SY:.

```
.D 44=2000
.SAVE SY:PRAM 1000-5777
```

The SET command changes device handler characteristics and certain system configuration parameters.



In the command syntax illustrated above, *physical-device-name* represents the device handler whose characteristics you need to modify.

See Table 3–1 in this manual for a list of the standard RT–11 permanent device names. The argument *item* represents a system parameter that you need to modify. The system items you can change include error handling (SET ERROR) and wildcard handling (SET WILD). Table 4–13 lists the devices and items you can modify, as well as the valid conditions for these devices and items. If you set more than one condition for a device, separate the conditions with commas. With the exception of the SET TT, SET USR, and SET item commands, the SET command locates the file SY:device.SYS and permanently modifies it. The SET commands are valid for all three RT–11 monitors unless otherwise specified. They permanently modify the device handlers (except where noted); this means that the conditions remain set even across a reboot. For those SET commands that do not permanently modify the device handlers, the conditions return to the default setting after a reboot. To make these settings appear permanent, include the appropriate SET commands in your system's startup indirect command file (see Section 4.3.3). The command you enter must be completely valid for the modification to take place.

NOTE

If a handler (except for TT:) is already loaded when you issue a SET command for it, you must unload the handler and install a fresh copy from the system device for the modification to have an effect on execution.

The colon (:) after each device name is optional.

Figure 4–2: Format of a 12–bit Binary Number

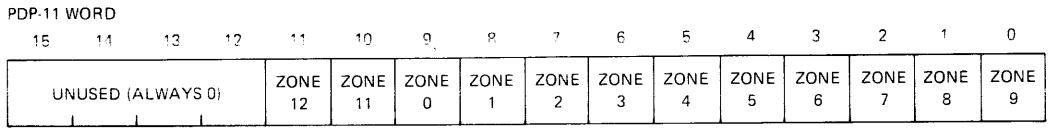


Table 4–13: SET Device Conditions and Modification

Device or Item	Condition	Modification
CR:	CODE = n	Modifies the card reader handler to use either the DEC 026 or DEC 029 card codes. The argument *n* must be either 26 or 29. The default value is 29.
CR:	CRLF	Appends a carriage return/line feed combination to each card image. This is the normal mode.
CR:	NOCRLF	Transfers each card image without appending a carriage return/line feed combination. The default is CRLF.
CR:	HANG	Waits for you to make a correction if the reader is not ready at the start of a transfer. This is the normal mode.
CR:	NOHANG	Generates an immediate error if the device is not ready at the start of a transfer. The handler waits (regardless of how the condition is set) if the reader is not ready at some point during a transfer (that is, the input hopper is empty, but an end-of-file card has not been read). The default is HANG.
CR:	IMAGE	Causes each card column to be stored as a 12-bit binary number, one column per word. The CODE option has no effect in IMAGE mode. Figure 4–2 illustrates the format of the 12-bit binary number. This format allows the system to read binary card images. It is especially useful if you use a special encoding of punch combinations. Mark-sense cards can be read in this mode. The default is NOIMAGE.
CR:	NOIMAGE	Allows the normal translation (as specified by the CODE option) to take place. The system packs data one column per byte. It translates invalid punch combinations into the error character, ASCII backslash (\), which is octal code 134. This is the normal mode.
CR:	TRIM	Removes trailing blanks from each card that the system reads. You should not use TRIM and NOCRLF together because card boundaries become difficult to read. TRIM is the normal mode.
CR:	NOTRIM	Transfers a full 80 characters per card. The default is TRIM.
CT:	RAW	Performs a read-after-write check for every record written. The system retries if an output error occurs. If three retries fail, the system indicates an output error. The default is NORAW.
CT:	NORAW	Writes every record directly without reading it back for verification. This setting significantly increases transfer rates at the risk of increased error rates. This is the normal mode.
DD:	VECTOR = n	Modifies the DECtape II handler to use *n* as the vector address for the first DECtape II controller (*n* is an octal number). This option, and the next three, enable you to set vector and Control and Status Register (CSR) values in the handler itself, without having to modify the handler source code and reassemble. Use these options if you have installed the DECtape II controller(s) at nonstandard addresses.

(continued on next page)

Table 4–13: SET Device Conditions and Modification (Cont.)

Device or Item	Condition	Modification
DD:	CSR=n	Modifies the DECtape II handler to use *n* as the CSR address for the first DECtape II controller. When you use this option, the system prints a message that indicates where to patch the DECtape II handler bootstrap, if you want to use a DECtape II as your system volume.
DD:	VEC2=n	Modifies the DECtape II handler to use *n* as the vector for the second DECtape II controller. This option is valid only if you create the DECtape II dual controller handler (through system generation).
DD:	CSR2=n	Modifies the DECtape II handler to use *n* as the CSR address for the second DECtape II controller. This option is valid only if you create the DECtape II dual controller handler (through system generation).
EDIT	EDIT	Invokes the text editor EDIT with the keyboard monitor EDIT command. This is the normal mode. The system returns to this condition after a reboot.
EDIT	KED	Invokes the Keypad Editor (KED). For more information on the Keypad Editor, see the *PDP–11 Keypad Editor User's Guide*. This condition is valid only for VT100 terminals. The system returns to EDIT EDIT after a reboot.
EDIT	K52	Invokes the Keypad Editor (K52); valid if your terminal is a VT52. For more information on the Keypad Editor, see the *PDP–11 Keypad Editor User's Guide*. The system returns to EDIT EDIT after a reboot.
EDIT	TECO	Invokes the text editor TECO with the keyboard monitor EDIT command. The default is EDIT. The system returns to that condition after a reboot.
ERROR	WARNING	Causes indirect command files and keyboard monitor commands to abort if warnings, errors, or severe or fatal errors occur. See SET ERROR ERROR, which is the default setting. Warning error messages contain the -W- characters. The system returns to that condition after a reboot.
ERROR	ERROR	Causes indirect command files and keyboard monitor commands that perform multiple operations (such as EXECUTE, which combines assembling, linking, and running) to abort if errors or severe or fatal errors occur. This setting causes indirect files and keyboard monitor commands to abort on MACRO assembly errors. An example of an error is an undefined symbol in an assembly. An example of a severe error is a device that is write-locked when the system attempts to write to it. If either condition occurs, the indirect command file or keyboard monitor command aborts the next time the monitor gets control of the system. Error error messages contain the -E- characters. This is the normal setting. The system returns to this condition after a reboot.

(continued on next page)

Table 4–13: SET Device Conditions and Modification (Cont.)

Device or Item	Condition	Modification
ERROR	SEVERE	Causes indirect command files and keyboard monitor commands to abort only if severe or fatal errors occur. Severe error messages contain the -F- characters. See SET ERROR ERROR, which is the default setting. The system returns to that condition after a reboot.
ERROR	NONE	Allows indirect command files and keyboard monitor commands to continue to execute even though they contain significant errors. Most monitor fatal errors still cause the indirect command file or keyboard monitor command to abort. Fatal errors that always abort indirect command files contain the -U- characters in the error messages. See SET ERROR ERROR, which is the default setting. The system returns to that condition after a reboot.
LP:	CR	Sends carriage returns to the printer. To allow overstriking on the printer, use this condition for any FORTRAN program that uses formatted input and output. Use CR also for any LS11 or LP05 line printer to prevent loss of the last line in the buffer. LP NOCR is the normal mode.
LP:	NOCR	Prevents the system from sending carriage returns to the printer. This setting produces a significant increase in printing speed on LP11 printers, where the line printer controller causes a line feed to perform the functions of a carriage return. This is the default setting.
LP:	CSR=n	Modifies the line printer handler to use *n* as the Control and Status Register (CSR) address for the line printer controller. The value you supply must be an octal word address not less than 160000. This option enables you to set a special CSR value in the line printer handler itself, without having to modify and reassemble the handler source code. Use this option if you have installed the line printer controller at a nonstandard address.
LP:	CTRL	Passes all characters, including nonprinting control characters, to the printer. Use this condition to pass the bell character to the LA180 printing terminal. You can use this mode for LS11 line printers. (Other line printers print a space for a control character.) The default is NOCTRL.
LP:	NOCTRL	Ignores nonprinting control characters. This is the normal mode.
LP:	FORM	Declares that the line printer has hardware form feeds, causing the line printer handler to send form feeds to the controller. When you use this option, the line printer handler sends the form feed character to the printer each time the handler encounters a form feed. This is the default setting.

(continued on next page)

Table 4–13: SET Device Conditions and Modification (Cont.)

Device or Item	Condition	Modification
LP:	NOFORM	Causes the line printer handler to simulate hardware form feeds by sending one or more line feeds to the printer. When you use this setting, you must also use the LENGTH = n setting and position the paper at the top of a form (that is, at the page perforation) before you start to use the printer. Using the NOFORM condition is useful if you are using a preprinted form that has a nonstandard length. You must use this setting if your printer does not accommodate form feeds. FORM is the default setting.
LP:	FORM0	Issues a form feed before a request to print blocks 0. This is the normal mode.
LP:	NOFORM0	Turns off FORM0 mode, which is the default.
LP:	HANG	Waits for you to make a correction if the line printer is not ready or is not ready at some point during printing. If you expect output from the line printer and the system does not respond or appears to be idle, check to see if the line printer is powered on and ready to print. This is the normal mode.
LP:	NOHANG	Generates an immediate error if the line printer is not ready. The default is HANG.
LP:	LC	Allows the system to send lower-case characters to the printer. Use this condition if your printer has a lower-case character set. The default is NOLC.
LP:	NOLC	Translates characters in lower case to upper case before printing. This is the normal mode.
LP:	LENGTH = n	Causes the line printer to use *n* as the number of lines per page. The default number of lines per page is 66. Use this option with the NOFORM and SKIP = n settings.
LP:	SKIP = n	Causes the line printer handler to send a form feed to the printer when it comes within n lines of the bottom of a page. Use this setting to prevent the printer from printing over page perforations. The value you supply for *n* should be an integer from 0 to the maximum number of lines on the paper. If you set SKIP = 0, the handler sends lines to the printer regardless of the position of the paper. To disable this condition, set SKIP = 0. When you use this setting, you must also use the LENGTH = n setting.
LP:	TAB	Sends TAB characters to the line printer. The default is NOTAB.
LP:	NOTAB	Expands TAB characters by sending multiple spaces to the line printer. This is the normal mode.
LP:	VECTOR = n	Modifies the line printer handler to use *n* as the vector of the line printer controller. The value you supply for *n* must be an even octal address below 500. This option enables you to set a special vector value in the line printer handler itself, without having to modify the handler source code and reassemble. Use this option if you have installed the line printer controller at a nonstandard address.

(continued on next page)

Table 4–13: SET Device Conditions and Modification (Cont.)

Device or Item	Condition	Modification
LP:	WIDTH=n	Sets the line printer width to *n*, where *n* is a decimal integer between 30 and 255, inclusive. The system ignores any characters that print past column *n*. The default is 132.
LS:	CR	Sends carriage returns to the printer. To allow overstriking on the printer, use this condition for any FORTRAN program that uses formatted input and output. (Use CR also for any LS11 or LP05 line printer to prevent loss of the last line in the buffer.) This is the normal mode.
LS:	NOCR	Prevents the system from sending carriage returns to the printer. This setting may produce a significant increase in printing speed on some line printers. Where the printer controller causes a line feed to perform the functions of a carriage return. The default is CR.
LS:	CSR=n	Modifies the line printer handler to use *n* as the Control and Status Register (CSR) address for the printer controller. The value you supply for *n* must be an octal word address not less than 160000. This option enables you to set a special CSR value in the printer handler itself, without having to modify the handler source code and reassemble. Use this option if you have installed the printer controller at a nonstandard address.
LS:	CTRL	Passes all characters, including nonprinting control characters, to the printer. Use this condition to pass the bell character to the LA180 printing terminal. The default is NOCTRL.
LS:	NOCTRL	Ignores nonprinting control characters. This is the normal mode.
LS:	FORM	Declares that the line printer has hardware form feeds, causing the line printer handler to send form feeds to the controller. When you use this option, the line printer handler sends the form feed character to the printer each time the handler encounters a form feed. This is the default setting.
LS:	NOFORM	Causes the line printer handler to simulate hardware form feeds by sending one or more line feeds to the printer. When you use this setting, you must also use the LENGTH=n setting and position the paper at the top of a form (that is, at the page perforation) before you start to use the printer. Using the NOFORM condition is useful if you are using a preprinted form that has a nonstandard length. You must use this setting if your printer does not accommodate form feeds. FORM is the default setting.
LS:	FORM0	Issues a form feed before a request to print block 0. This is the normal mode.
LS:	NOFORM0	Turns off FORM0 mode. The default is FORM0.
LS:	HANG	Waits for you to make a correction if the line printer is not ready or becomes not ready during printing. If you expect output from the printer and the system does not respond or appears to be idle, check to see if the printer is powered on and ready to print. This is the normal mode.

(continued on next page)

Table 4–13: SET Device Conditions and Modification (Cont.)

Device or Item	Condition	Modification
LS:	NOHANG	Generates an immediate error if the printer is not ready. The default setting is HANG.
LS:	LC	Allows the system to send lower-case characters to the printer. Use this condition if your printer has a lower-case character set. The default is NOLC.
LS:	NOLC	Translates lower-case characters to upper-case before printing. This is the normal mode.
LS:	LENGTH = n	Causes the printer to use *n* as the number of lines per page. The default number of lines per page is 66. Use this option with the NOFORM and SKIP = n settings.
LS:	SKIP = n	Causes the line printer handler to send a form feed to the printer when it comes within n lines of the bottom of a page. Use this setting to prevent the printer from printing over page perforations. The value you supply for *n* should be an integer from 0 to the maximum number of lines on the paper. If you set SKIP = 0, the handler sends lines to the printer regardless of the position of the paper. To disable this condition, set SKIP = 0. When you use this setting, you must also use the LENGTH = n setting.
LS:	TAB	Sends TAB characters to the printer. The default is NOTAB.
LS:	NOTAB	Expands TABS by sending multiple spaces to the printer. This is the normal mode.
LS:	VECTOR = n	Modifies the printer handler to use *n* as the vector of the line printer controller. The value you supply for *n* must be an even octal address below 500. This option enables you to set a special vector value in the line printer handler itself, without having to modify the handler source code and reassemble. Use this option if you have installed the printer controller at a nonstandard address.
LS:	WIDTH = n	Sets the printer to width *n*, where *n* is a decimal integer between 30 and 255, inclusive. The system ignores any characters that print past column *n*. The default is 132.
MM:	DEFALT = 9	Returns to default settings for 9-track tape. The 9-track defaults are: DENSE = 809 ODDPAR NODUMP
MM:	DENSE = [800 or 809 or 1600]	Sets density for the 9-track tape handler. Do not alter the density setting within a volume. A density setting of 1600 bits per inch (BPI) automatically sets parity to odd. The valid density settings for 9-track tape are: 800 BPI 1600 BPI
MM:	ODDPAR	Sets parity to odd for 9-track tape. DIGITAL recommends this setting.

(continued on next page)

Table 4–13: SET Device Conditions and Modification (Cont.)

Device or Item	Condition	Modification
MM:	NOODDPAR	Sets parity to even for 9-track tape. DIGITAL does not recommend this setting for normal operation, and provides it only for compatibility with other systems.
MT:	DEFALT = [7 or 9]	Returns to default settings for 7- or 9-track tape. The 7-track defaults are: DENSE = 807 ODDPAR DUMP The 9-track defaults are: DENSE = 809 ODDPAR NODUMP The default setting is DEFALT = 9.
MT:	DENSE = [200 or 556 or 807 or 809]	Sets density for 7- or 9-track tape. 807 represents 800 BPI for 7-track tape; 809 represents 800 BPI for 9-track tape. Do not alter the density within a tape volume. You must set density to 807 for 7 track tape if you want dump mode. The valid density settings for 7 and 9 track tape are: 7-track: 200 BPI 556 BPI 800 BPI 800 BPI Dump 9-track: 800 BBI
MT:	DUMP	Writes bytes to 7-track tape. You must also set density to 807.
MT:	ODDPAR	Sets parity to odd for 7- or 9-track tape. DIGITAL recommends this setting.
MT:	NOODDPAR	Sets parity to even for 7- or 9-track tape. DIGITAL does not recommend this setting for normal operation, and provides it only for compatibility with other systems.
TT:	CONSOL = n	Directs the system to use the terminal whose logical unit number you specify as the console terminal. The terminal whose logical unit number you specify must not be currently attached by the foreground or any system job. To use this setting, you must have a multi-terminal configuration. The system returns to this default after a reboot. You cannot use this setting for a remote line.
TT:	CRLF	Issues a carriage return/line feed combination on the console terminal whenever you attempt to print past the right margin. You can change the margin with the WIDTH command. This is the normal mode. This setting is invalid with a non-multi-terminal SJ monitor. The system returns to this condition after a reboot.
TT:	NOCRLF	Takes no special action at the right margin. This setting is invalid with a non-multi-terminal SJ monitor. The default is CRLF. The system returns to that condition after a reboot.

(continued on next page)

Table 4–13: SET Device Conditions and Modification (Cont.)

Device or Item	Condition	Modification
TT:	FB	Treats CTRL/B and CTRL/F (and CTRL/X in system job monitors) as background and foreground program control characters and does not transmit them to your program. This is the normal mode. This setting is not valid for the SJ monitor. The system returns to this condition after a reboot.
TT:	NOFB	Causes CTRL/B and CTRL/F (and CTRL/X in system job monitors) to have no special meaning. Issue SET TT: NOFB to KMON, which runs as a background job, to disable all communication with the foreground or system job. To enable communication with the foreground job, issue the command SET TT FB. This setting is not valid for the SJ monitor. The default is FB. The system returns to that condition after a reboot.
TT:	FORM	Indicates that the console terminal is capable of executing hardware form feeds. This setting is invalid with a non-multi-terminal SJ monitor.
TT:	NOFORM	Simulates form feeds by generating eight line feeds. This setting is not valid for the non-multi-terminal SJ monitor. This is the normal mode. The system returns to this condition after a reboot.
TT:	HOLD	Enables the Hold Screen mode of operation for the VT50, VT52, and VT61 terminals. The command has no effect on any other terminals, but it can cause a left square bracket ([) to print. This setting is valid for all monitors. NOHOLD is the default setting. The system returns to that condition after a reboot.
TT:	NOHOLD	Disables the Hold Screen mode of operation for the VT50 terminal. The command has no effect on any other terminal, but it can cause a backslash (\) to print. This setting is valid for all monitors. The default is NOHOLD. The system returns to that condition after a reboot.
TT:	PAGE	Treats CTRL/S and CTRL/Q characters as terminal output hold and unhold flags and does not transmit them to your program. You must use this setting if you are using a VT100 terminal. This setting is not valid for the non-multi-terminal SJ monitor. This is the normal mode. The system returns to this condition after a reboot.
TT:	NOPAGE	Causes CTRL/S and CTRL/Q to have no special meaning. This setting is not valid for the non-multi-terminal SJ monitor. The default is PAGE. The system returns to that condition after a reboot.
TT:	QUIET	Prevents the system from echoing lines from indirect files. The default is NOQUIET. The system returns to that condition after a reboot.
TT:	NOQUIET	Echoes lines from indirect files. This is the default mode. The system returns to this condition after a reboot.

(continued on next page)

Table 4–13: SET Device Conditions and Modification (Cont.)

Device or Item	Condition	Modification
TT:	SCOPE	Echoes RUBOUT characters as backspace-space-backspace. Use this mode if your console terminal is a VT50, VT05, VT52, VT55, VT61, VT100, or if GT ON is in effect. The default is NOSCOPE. The system returns to that condition after a reboot. Note that you delete TAB characters by typing a single RUBOUT or DELETE, even though the cursor does not move back the correct number of spaces. This is a restriction in SCOPE modes.
TT:	NOSCOPE	Echoes RUBOUT characters by enclosing the deleted characters in backslashes. This is the normal mode. The system returns to this condition after a reboot.
TT:	TAB	Indicates that the console terminal is capable of executing hardware tabs. This setting is not valid for the non-multi-terminal SJ monitor. The default is NOTAB. The system returns to that condition after a reboot.
TT:	NOTAB	Simulates tab stops every eight positions. Many terminals supplied by DIGITAL have hardware tabs. This setting is not valid for the non-multi-terminal SJ monitor. This is the normal mode. The system returns to this condition after a reboot.
TT:	WIDTH = n	Sets the terminal width to *n*, where *n* is an integer between 30 and 255. The system initially sets the width to 80. This setting is not valid for the non-multi-terminal SJ monitor. (See SET TT CRLF.) The system returns to 80 after a reboot.
USR	SWAP	Allows the background job to place the user service routine (USR) in a swapping state. This setting is not valid for the XM monitor. This is the normal mode for FB and SJ monitors. The system returns to this condition after a reboot.
USR	NOSWAP	Prevents the background job from placing the USR in a swapping state. This setting is not valid for the XM monitor. The default is SWAP for FB and SJ monitors. The system returns to that condition after a reboot.
WILD	EXPLICIT	Causes the system to recognize file specifications exactly as you type them. If you omit a file name or a file type in a file specification the system does not automatically replace the missing item with an asterisk (*). Wildcards are described in Section 4.2 of this manual. The default is IMPLICIT. The system returns to that condition after a reboot.
WILD	IMPLICIT	Causes the system to interpret missing fields in file specifications as asterisks (*). Wildcards are described in Section 4.2 of this manual. Table 4–2 shows how the system interprets commands that have missing fields. This is the normal mode. The system returns to this condition after a reboot.

The following examples illustrate the SET command. This command allows the system to send lower-case characters to the printer:

```
.SET LP LC
```

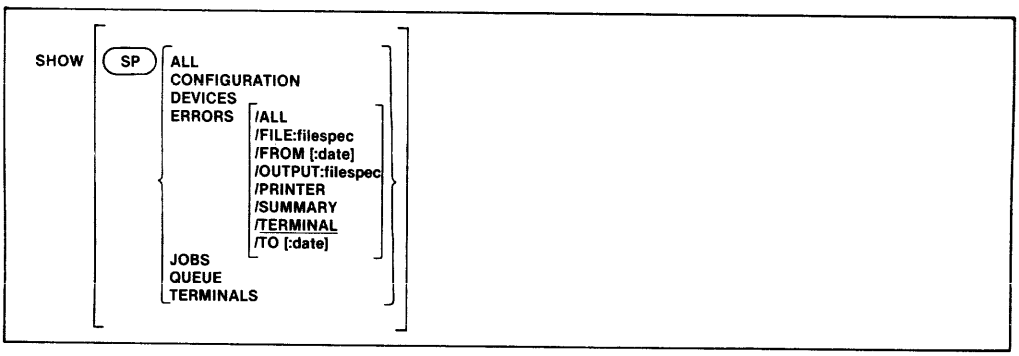
The next command sets the system wildcard default to implicit.

```
.SET WILD IMPLICIT
```

As a result of this command the system inserts an asterisk in place of a missing file name or file type in a file specification. See Table 4–2 for a list of these commands.

SHOW

The SHOW command prints information about your RT–11 system on the console terminal.



The information includes hardware configuration, monitor version, special features in effect, device names and logical device name assignments, terminal characteristics for terminals currently active on a multi-terminal system, and device handler status. If you are running the Error Logger or QUEUE, the SHOW command can provide information on errors and the update status of files waiting to be sent to an output device.

If you specify SHOW without an option, SHOW displays your system's device assignments. The devices the system lists are those known by the RT–11 monitor currently running in memory. This list reflects any additions or deletions you have made with the INSTALL and REMOVE commands. The listing also includes additional information about particular devices. The informational messages and their meanings are:

Message	Indicated Condition
(RESORC) or =RESORC	The device or unit is assigned to the background job RESORC (for FB and XM monitors only).
(FORE) or =FORE	The device or unit is assigned to the foreground job (for FB and XM monitors only and monitors without system job support).
(jobname) or =jobname	The device or unit is assigned to the system or foreground job (for FB and XM monitors that have system job support), where *jobname* represents the name of the system or foreground job.
(Loaded)	The handler for the device has been loaded into memory with the LOAD command.
(Resident)	The handler for the device is included in the resident monitor.

=logical-device-name(1), logical-device-name(2)... ,logical-device-name(n)	The device or unit has been assigned the indicated logical device names with the ASSIGN command.
xx free slots	The last line tells the number of unassigned, or free, device slots.

The following example was created under an FB monitor that has system job support. It shows the status of all devices known to the system.

```
.SHOW
TT   (Resident)
RK   (Resident)
  RK1    = SY, DK, OBJ, SRC, BIN
  RK2    = LST, MAP
MQ   (Resident)
DL   (Loaded)
DM
DX   (Loaded)
  DX0:   (MYPROG)
  DX1:   (RESORC)
LP:  (Loaded=QUEUE)
MT
CT
5 free slots
```

The listing shows first that TT and RK are resident in memory. The other device handlers known to the system are MQ, DL, DM, DX, LP, MT, and CT. There are five free slots in the table. RK0: has the logical names SY, DK, OBJ, SRC, and BIN. RK1: has the logical names LST and MAP. The DX handler is loaded and device DX0: belongs to the foreground job, MYPROG. The LP: handler is loaded and belongs to the system job, QUEUE.

The options for the SHOW command follow.

ALL This option is a combination of CONFIGURATION, DEVICES, JOBS, and TERMINALS, in that order. The ALL option also tests the device assignments.

CONFIGURATION This option displays the monitor version number and patch level, the monitor SET options in effect, the hardware configuration, and the special features in effect (if any). The listing varies, of course, depending on which monitor and which hardware system you are using.

First, the listing always shows the version number and patch level of the currently running monitor.

Next, information about the monitor is displayed. The first line indicates the device from which the system was bootstrapped. The next line prints the resident monitor's base address, in octal. Then the listing shows whether the user service routine (USR) is set to SWAP or NOSWAP. Another line prints out if a foreground job is loaded. The listing shows whether TT is set QUIET or NOQUIET, and whether the indirect file abort level is set to NONE, WARNING, ERROR, or SEVERE. The indirect file nesting depth prints out as a decimal number.

Next, the listing displays the system hardware configuration. It lists the processor type, which can be one of the following:

LSI 11
PDT 130/150
PDP 11/04
PDP 11/05,10
PDP 11/15,20
PDP 11/23
PDP 11/34
PDP 11/35,40
PDP 11/45,50,55
PDP 11/60
PDP 11/70

A separate line prints out for each of the following items that is present on your system:

FP11 Hardware Floating Point Unit
Commercial Instruction Set (CIS)
Extended Instruction Set (EIS)
Floating Instruction Set (FIS)
KT11 Memory Management Unit
Parity Memory
Cache Memory

If you have graphics hardware (VT11 or VS60), another line is printed out to indicate it. The clock frequency (50 or 60 cycles) prints next, followed by a line for the KW11–P programmable clock, if there is one on your system.

Finally, the listing either shows that there are no special features in effect, or it lists the appropriate features from the following list:

Device I/O time-out support
Error logging support
Multi-terminal support
Memory parity support
SJ timer support
System job support

The following example was created on a PDP 11/23 processor:

```
.SHOW CONFIGURATION

RT-11FB(S)  V04.00

Booted from RK0:
Resident Monitor base is 137500 (48960.)
USR is set SWAP
TT is set NOQUIET
Indirect file abort level is ERROR
Indirect file nesting depth is 3

PDP 11/23
60 Cycle System Clock
Error logging support
Device I/O time-out support
Memory parity support
```

DEVICES This option displays the RT–11 device handlers, their status, and their vectors. The messages for handler status are as follows:

Installed
Not installed
-Not installed (the handler special features do not match those of the monitor)
nnnnnn (load address of handler)
Resident

The following example uses SHOW DEVICES.

```
.SHOW DEVICES
Device       Status          Vector

  DX       Installed        264
  RK       Resident         220
  RF       Not installed    204
  DT       Installed        214
  LP       Installed        200
  CR       Not installed    230
  NL       Installed        000
  PC       Installed        070 074
  CT       Installed        260
  DS       Installed        204
  DM       Installed        210
  DL       Installed        330
  DP       Installed        254
  DY       Installed        270
  MT       Installed        224
  MM       Not installed    224
```

In the preceding example, note that the PC handler has two vectors. One is for the paper tape reader and the other is for the paper tape punch. Because of its special format, the TT handler is never listed.

JOBS This option displays data about the jobs that are currently loaded. This option also tells the following:

- the job name and number (if you have not enabled system job support on your monitor, the foreground job name appears as FORE, and its priority is 1)
- the console the job owns (if a non-multi-terminal monitor, this space is blank)
- the priority level of the job
- the job's running state (running, suspended, or done (but not unloaded))
- the low and high memory limits of the job
- the start address of the job's impure area

The example that follows displays data about currently running jobs:

```
.SHOW JOBS

Job   Name   Console Level State    Low    High   Impure
14    QUEUE      0     6   Suspend 116224 130306 115254
 0    RESORC     0     0   Run     000000 115210 134126
```

ERRORS The SHOW ERRORS command is valid only if you have error logging enabled on your monitor. For a complete description of the error logger and directions on how to start it, see Chapter 19, Error Logging. Note that the error logger is a special feature, available only through the system generation process. Because the error logger compiles statistics on each I/O transfer that occurs, in addition to hardware errors that occur, it is a good idea to enable error logging on a spare system volume that you use only when you want to compile error statistics.

The SHOW ERROR command invokes ERROUT, one of the three programs in the error logging package that runs as a background job. ERROUT creates reports on the I/O and error statistics the error logger compiles, and can print the reports at the terminal, line printer, or store the reports in a file you specify. For complete descriptions of the reports ERROUT creates, see Chapter 19, Error Logging.

ERRORS <RET>
: prints a full report on each I/O transfer that has occurred in addition to each I/O, memory parity, and cache memory error that has occurred.

ERRORS/ALL
: same as SHOW ERRORS <RET>

ERRORS/FILE:filespec
: prints a full I/O transfer and error report from the file you specify. The file you specify must be of the same format that the error logger uses for its statistics compilations.

ERRORS/FROM[:date]
: prints a full I/O transfer and error report for errors that occurred starting from the date you specify. Enter the date as *dd:mmm:yy*, where

  dd is a two-digit date (decimal)

  mmm is the first three characters of a month

  yy is a two-digit year

ERRORS/TO[:date]
: prints a full I/O transfer and error report for errors that occurred up to the date you specify.

ERRORS/OUTPUT:filespec
: enters the I/O transfer and error report in the output file you specify. This is useful if you want to save the error logging reports.

ERRORS/PRINTER	prints the I/O transfer and error report at the line printer.
ERRORS/SUMMARY	prints a summary error report at the terminal. The summary error report lists only the errors that occurred, not the successful I/O transfers.
ERRORS/TERMINAL	prints the I/O transfer and error report at the terminal. /TERMINAL is the default setting.

QUEUE Use the SHOW QUEUE command to get a listing of the contents of the queue. This option is invalid if you are not running QUEUE (see Chapter 20, Queue Package). The listing shows the output device, job name, input files, job status, and number of copies for each job that is queued. The sample command line that follows lists the current contents of the queue.

```
.SHOW QUEUE
DEVICE     JOB      STATUS     COPIES    FILES

 LP0:      LAB2       P          1       PASS3 .LST
                                 2       PASS4 .LST
                                 2       PASS5 .LST
 LP0:      HODG       Q          3       MESMAN.DOC
 DT1:      JUDITH     Q          2       PART1 .DOC
                                 2       PART2 .DOC
 MT1:      SZYM       Q          1       REFMAN.TXT
 LP0:      JOYCE      Q          1       SSM   .DOC
                                 1       DOCPLN.DOC
```

The job STATUS column prints a *P* if the job is currently being output, an *S* if the job being output is suspended, or a *Q* if the job is waiting to be output. If you have a large lineup of files, and your console is a video terminal, you can use the CTRL/S and CTRL/Q commands to control the scrolling of the listing.

TERMINALS This option indicates the status of and special features in effect for currently active terminals on multi-terminal systems. If your system has only the console terminal, the following message prints:

```
No multi-terminal support
```

Multi-terminal support is a special feature; it is not part of the distributed RT–11 monitors.

If your system does have multi-terminal support, SHOW TERMINALS prints a table of the existing terminals and lists the following information:

Unit number	**(0-15)**
Owner:	Background, foreground, system job owner, or none
Type:	Local Remote (dial-up) Console

(continued on next page)

Unit number **(0-15)**

S-console (shared by background and foreground or system job)
Is attached to another job (the foreground)

Interface type: DL DZ

Width: (width in characters, up to 255)

SET options in effect:
TAB
CRLF
FORM
SCOPE

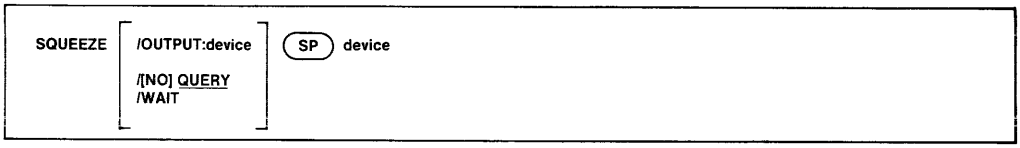
Line speed: (baud rate if DZ; not applicable if DL)

The following example shows the terminal status of an RT–11 system.

```
.SHOW TERMINALS
Unit  Owner      Type       WIDTH TAB CRLF FORM SCOPE SPEED

  0   RESORC  S-Console DL   132  No  Yes   No   No    N/A
  1   FORE    Local     DL    80  Yes No    No   Yes   N/A
```

SQUEEZE

The SQUEEZE command consolidates all unused blocks into a single area on the device you specify and consolidates the directory entries on the device.



In the command syntax illustrated above, *device* represents the block replaceable volume to be compressed. To perform a squeeze operation, the system moves all the files to the beginning of the device you specify, producing a single unused area after the group of files. The squeeze operation does not change the bootstrap blocks of a device. The system prints a confirmation message before it performs the squeeze operation. You must type Y followed by a carriage return to execute the command.

The squeeze operation does not move files with .BAD file types. This feature prevents you from reusing bad blocks that occur on a disk. The system inserts files before and after .BAD files until the space between the last file it moved and the .BAD file is smaller than the next file to be moved. Note that you should use the SQUEEZE command when you get a *directory full* error, even if there is still space remaining on the volume.

If you perform a squeeze operation on the system device, the system automatically reboots the running monitor when the compress operation completes. This reboot takes place in order to prevent system crashes that might occur when the monitor file or handler files are moved. The system volume cannot be squeezed if a foreground or system job is loaded.

The options for the SQUEEZE command follow.

/OUTPUT:filespec Use this option to transfer all the files from the input device to the output device in compressed format, an operation that leaves the input device unchanged. The output device must be an initialized block replaceable volume. (Use the INITIALIZE command to do this.) Note that the system does not query you for confirmation before this operation proceeds. If the output device is not initialized, the system prints an error message and does not execute the command. Note that /OUTPUT does not copy boot blocks; you must use the COPY/BOOT command to make the output volume bootable. The following example transfers all the files from RK0: to RK1: in compressed format, leaving RK0: unchanged.

```
.SQUEEZE/OUTPUT:RK1: RK0:
```

/QUERY This option causes the system to print a confirmation message before it executes a squeeze operation. You must respond by typing a Y followed by a carriage return for execution to proceed. This is the default operation. /QUERY is invalid with the /OUTPUT option.

/NOQUERY Use this option to suppress the confirmation message that prints before a squeeze operation executes. The following command compresses all the files on device DT1: and does not query.

```
.SQUEEZE/NOQUERY DT1:
```

/WAIT This option is useful if you have a single-disk system. When you use /WAIT, the system initiates the SQUEEZE operation, but then pauses and waits for you to mount the volume you want to squeeze. When the system pauses, it prints *Mount input volume in <device>; Continue?*, where *<device>* represents the device into which you mount the volume. When the volume is mounted, type Y followed by a carriage return.

The following sample command line squeezes an RK05 disk:

```
.SQUEEZE/WAIT RK0:
RK0:/Squeeze; Are you sure? Y
Mount input volume in RK0:; Continue? Y
Mount system volume in RK0:; Continue? Y
```

Note that the system may repeat the *Mount input volume—Mount output volume* cycle several times to complete the SQUEEZE operation.

SRUN

The SRUN command initiates system jobs.

In the command syntax illustrated above, *filespec* represents the program to be executed. Because this command runs a system job, it is valid only for FB and XM monitors that have system job support — a special feature enabled through the system generation process.

You can run up to six system jobs simultaneously, in addition to the foreground and background jobs. If you attempt to run a system job that is already active, an error message prints on the terminal.

Note that when you issue the SRUN command, the monitor assumes a .SYS file type. If you want to issue the SRUN command for a program that has a .REL file type, either enter the file type with the file name (for example, SRUN QUEUE.REL), or rename the file so it has a .SYS file type.

In an XM monitor, you can use the SRUN command to run a virtual .SAV image program. You must type the file type explicitly.

The options that you can use with SRUN follow.

/BUFFER:n Use this option to reserve space in memory over the actual program size. The argument *n* represents the number of octal words of memory to allocate. You must use this option to run any FORTRAN program as a system job. If you use this option for a virtual job linked with the /V option (or /XM), the system ignores /BUFFER because the system provides a buffer in extended memory.

/LEVEL:n Use this option to assign an execution priority level to the job, where *n* can be 1, 2, 3, 4, 5, or 6. If you attempt to assign the same priority level to two system jobs, an error message prints at the terminal. If omitted, the priority level defaults to the highest level that is thus far unassigned.

/NAME:logical-jobname Use this option to assign a logical job name to a program. This is the name that programmed requests and SYSLIB calls use to reference a system job. If you attempt to assign the same logical job name to two system jobs, an error message prints at the terminal. If you do not specify a logical job name, the system assumes the file name of the program.

/PAUSE Use this option to help you debug a program. When you type the carriage return as the end of the command string, the system prints the load address of your program and waits. By means of ODT, you can examine or modify the program before starting execution (see Chapter 21). You must use the RESUME command to restart the system job. The following com-

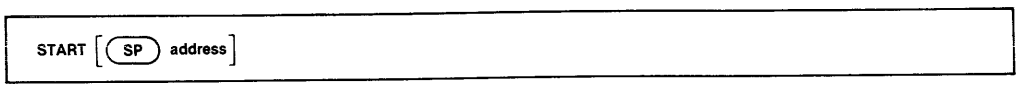
mand loads the program MFUNCT.SYS, prints the load address, and waits for a RESUME command to begin execution.

```
.SRUN MFUNCT/PAUSE
Loaded at 126556
.RESUME MFUNCT
```

/TERMINAL:n Use this option to change the console of the system job. Your system must have multi-terminal support — a special feature available only through system generation — before you can use it. The argument *n* represents a terminal logical unit number. By assigning a different terminal to interact with the system job, you eliminate the need for system, foreground, and background jobs to share the console terminal. Note that the original console terminal still interacts with the background job and with the keyboard monitor, unless you use the SET TT: CONSOL command to change this.

START

The START command initiates execution of the program currently in memory (loaded with the GET command) at the address you specify.



In the command syntax shown above, *address* is an even octal number representing any 16-bit address. If you omit the address or if you specify 0, the system uses the starting address that is in location 40. If the address you specify does not exist or is invalid for any reason, a trap to location 4 occurs and the monitor prints an error message. Note that this command is valid for background jobs only, and not for extended memory virtual jobs. The following command loads MYPROG.SAV into memory and begins execution.

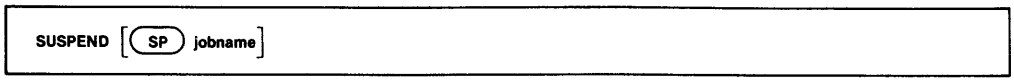
```
.GET MYPROG
.START
```

The next example loads MYPROG.SAV and ODT.SAV into memory, and begins execution at ODT's starting address.

```
.GET MYPROG
.GET ODT
.START 1222
 ODT V01.04
*
```

SUSPEND

The SUSPEND command temporarily stops execution of the foreground or system job.



If you have system job support enabled on your monitor, specify the name of the system or foreground job you wish to suspend. If you do not have system job support, then do not include an argument with the SUSPEND command. The SUSPEND command is not valid for the SJ monitor. The system permits foreground input and output that are already in progress to finish; however, it issues no new input or output requests and enters no completion routines (see the *RT–11 Programmer's Reference Manual* for a detailed explanation of completion routines). You can continue execution of the job by typing the RESUME command. The following command suspends execution of the foreground job that is currently running on a system that does not have system job support.

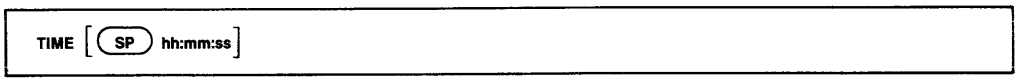
```
.SUSPEND
```

The next command suspends execution of the system job, QUEUE, that is currently running on a system that does have system job support.

```
.SUSPEND QUEUE
```

TIME

Use the TIME command to set the time of day or to display the current time of day.



In the command syntax shown above, *hh* represents hours (from 0 to 23); *mm* represents minutes (from 0 to 59) and *ss* represents seconds (from 0 to 59). The system keeps time on a 24-hour clock.

To enter the time of day, specify the time in the format described above. You should do this as soon as you bootstrap the system. The following example enters the time, 11:15:00 A.M.

```
.TIME 11:15
```

As this example shows, if you omit one of the arguments the system assumes 0.

To display the current time of day, type the TIME command without an argument, as this example shows.

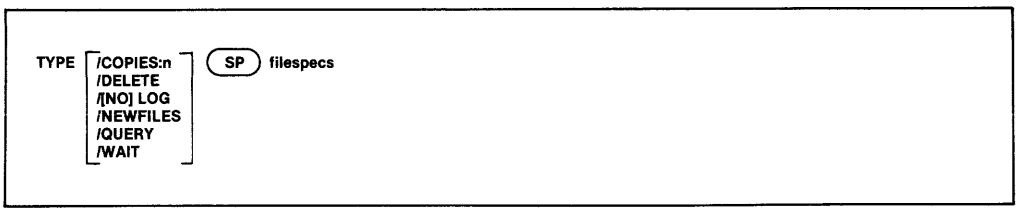
```
.TIME
11:15:01
```

When you install the standard RT–11 monitors, the clock rate is preset to 60 cycles. Consult the *RT–11 Installation and System Generation Guide* for information on setting the clock to a 50-cycle rate.

The FB and XM monitors automatically reset the time each day at midnight. The SJ monitor resets the time only if you select timer support during the system generation process.

TYPE

The TYPE command prints the contents of one or more files on the terminal.



In the command syntax illustrated above, *filespecs* represents the file or files to be typed. You can explicitly specify up to six files as input to the TYPE command. The system types the files in the order in which you specify them in the command line. You can also use wildcards in the file specification. In this case, the system types the files in the order in which they occur in the directory of the device you specify. If you specify more than one file, separate the files by commas. If you omit the file type for a file specification, the system assumes .LST. You can specify the entire command on one line, or you can rely on the system to prompt you for information. The TYPE command prompt is *Files?*.

The TYPE command options and examples follow.

/COPIES:n Use this option to list more than one copy of the file. The meaningful range of values for the decimal argument *n* is from 2 to 32 (1 is the default). The following command, for example, types three copies of the file REPORT.LST on the terminal.

```
.TYPE/COPIES:3 REPORT
```

/DELETE Use this option to delete a file after it is typed on the terminal. This option must appear following the command in the command line. The TYPE/DELETE operation does not ask you for confirmation before it executes. You must use /QUERY for this function. The following example types a BASIC program on the terminal, then deletes it from DX1:.

```
.TYPE/DELETE DX1:PROG1.BAS
```

/LOG This option prints on the terminal the names of the files that were typed by the current command. Normally, the system prints a log only if there is a wildcard in the file specification. If you specify /QUERY, the query messages replace the log, unless you specifically type /LOG/QUERY in the command line. The following example shows a TYPE command and the resulting log.

```
.TYPE/LOG OUTFIL.LST
 Files copied:
DK:OUTFIL.LST    to TT:
```

/NOLOG This option prevents a list of the copied files from printing on the terminal. You can use this option to suppress the log if you use a wildcard in the file specification.

/NEWFILES Use this option in the command line if you need to type only those files that have the current date. The following example shows a convenient way to type all new files after a session at the computer.

```
.TYPE/NEWFILES *.LST
 Files copied:
DK:REPORT.LST    to TT:
```

/QUERY If you use this option, the system requests confirmation before it performs the operation. /QUERY is particularly useful on operations that involve wildcards, when you may not be sure which files the system selected for an operation. Note that if you specify /QUERY in a TYPE command line that also contains a wildcard in the file specification, the confirmation messages printed on the terminal replace the log messages that would normally appea You must respond to a query message by typing Y (or anything that begins with y) and a carriage retun to initiate execution of a articular operation. The system interprets ay other response to mean NO and does not perform the specific operation.

```
.TYPE/QUERY/DELETE *.LST
 Files copied/deleted:
DK:OUTFIL.LST    to TT:? NO
DK:REPORT.LST    to TT:? Y
```

/WAIT This option is useful if you have a single-disk system. When you use this option, the system initiates the TYPE operation, but then pauses and waits for you to mount the volume on which you want the operation to take place. When the system pauses, it prints *Mount input volume in <device>; Continue?,* where *<device>* represents the device into which you mount the volume. When you have mounted the volume, type Y followed by a carriage return.

The following sample command types AJAX.DOC from an RK05 disk:

```
.TYPE/WAIT RK0:AJAX.DOC
Mount input volume in RK0:; Continue?Y
```

After the system has typed AJAX.DOC at the terminal, it issues the following prompt:

```
Mount system volume in RK0:;  Continue?
```

When you mount the system volume, and type a Y followed by a carriage return, you terminate the TYPE operation.

UNLOAD

The UNLOAD command removes previously loaded handlers from memory, thus freeing the memory space they occupied. It also removes terminated foreground or system jobs.

In the command syntax shown above, *device* represents the device handler to be unloaded. The colon that follows the device handler is optional with SJ, FB, and XM monitors and monitors that do not have system job support. You must include the colon if your system has system job support.

UNLOAD clears ownership for all units of the device type you specify. A request to unload the system device handler clears ownership for any assigned units for that device, but the handler itself remains resident. After you issue the UNLOAD command, the system returns any memory it frees to a free memory list. The background job eventually reclaims free memory. Note that if you interrupt an operation that involves magtapes or cassette, you must unload and then load (with the LOAD command) the appropriate device handler (MM, MT, MS, or CT).

The system does not accept an UNLOAD command while a foreground job is running if the foreground job owns any units of that device. This is because a handler that the foreground job needs might become nonresident. You can unload a device while a foreground job is running if none of its units belong to the foreground job.

A special function of this command is to remove a terminated foreground or system job and reclaim memory, since the system does not automatically return the space occupied by the foreground or system job to the free memory list. The following command unloads the foreground job and frees the memory it occupied. This command is valid only if the foreground job is not running.

```
.UNLOAD F
```

The following command unloads the system job QUEUE.

```
.UNLOAD QUEUE
```

The following command clears ownership of all units of RK:. If RK: is the system device, the RK handler itself remains resident.

```
.UNLOAD RK:
```

The next command releases the line printer and DECtape handlers and frees the area they previously held.

```
.UNLOAD LP:,DT:
```

Part III
Text Editing

You use an editor to create and modify textual material. Part III describes the RT–11 text editor, EDIT, and explains how to use it.

Chapter 5
Text Editor (EDIT)

The text editor (EDIT) is a program that creates or modifies ASCII source files for use as input to other system programs such as the MACRO assembler or the FORTRAN compiler. EDIT, which accepts commands you type at the terminal, reads ASCII files from any input device, makes specific changes, and writes on any output device. EDIT allows efficient use of VT11 or VS60 display hardware, if they are part of the system configuration.

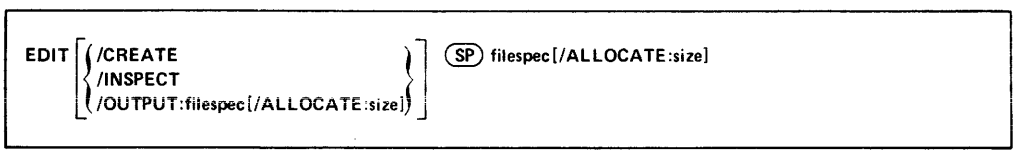
The editor considers a file to be divided into logical units called pages. A page of text is generally 50 to 60 lines long (delimited by form feed characters) and corresponds approximately to a physical page of a program listing. The editor reads one page of text at a time from the input file into its internal text buffers, where the page becomes available for editing. Editing commands can:

- Locate text to be changed
- Execute and verify changes
- List an edited page on the console terminal
- Output a page of text to the output file

Note that you cannot perform any edit operations on a protected file. To disable a file's protected status, see the RENAME command description.

5.1 Calling EDIT

You can call the text editor when you are at monitor level. The syntax of the command is:



See Section 4.4 for a description of the EDIT command and its options.

5.2 Modes of Operation

The editor operates in either command mode or text mode. In command mode the editor interprets all input you type on the keyboard as commands to perform some operation. In text mode the editor interprets all typed input

as text to replace, insert into, or append to the contents of the text buffer. You can use a special editing mode, called immediate mode, with the VT–11 display hardware. Section 5.7.2 describes this mode.

Immediately after being loaded into memory and started, the editor is in command mode. EDIT prints an asterisk at the left margin of the console terminal page to indicate that it is ready to accept a command. Terminate all commands by pressing the ESCAPE key twice in succession. Execution of commands proceeds from left to right. When EDIT encounters an error before beginning execution of a command string, it prints an error message followed by an asterisk at the beginning of a new line, indicating that it is still in command mode, that it is waiting for a command, and that execution of the illegal command has not occurred. You should retype the command correctly.

To enter text mode, type a command that must be followed by a text string. These commands insert, replace, exchange, or otherwise manipulate text. When you type one of these commands, EDIT recognizes all succeeding characters as part of the text string until it encounters an ESCAPE character. The ESCAPE terminates the text string and causes the editor to reenter command mode.

5.3 Special Key Commands

Table 5–1 lists the EDIT key commands. Type a control command by holding down the CTRL key while typing the appropriate character.

Table 5–1: EDIT Key Commands

Key	Explanation
ESCAPE, ALTMODE, or SEL	Echoes as $. A single ESCAPE terminates a text string. A double ESCAPE (two consecutive ESCAPEs) executes the command string. For example: `*GMOV A,B$-1D$$` The first ESCAPE ($) terminates the text object (MOV A,B) of the Get command. The double ESCAPE ($$) terminates the Delete command and causes execution of the entire statement with the result that the character B will be deleted.

CTRL/C	Echoes at the terminal as ^C. If EDIT encounters a CTRL/C as a command in command mode, it terminates execution and returns control to the monitor. You can restart the editor by typing R EDIT or REENTER in response to the monitor's prompt. If EDIT encounters a CTRL/C in a text object, the CTRL/C is included in the text object, as if it were just like any other character. If the editor is executing a lengthy command and you want to stop EDIT, type two CTRL/C commands in succession. This will abort the command, generate the *?EDIT-F-COMMAND ABORTED* error message, and return the editor to command mode. For example:
	`*I^C^C^C$$`
	`*^C$$`

(continued on next page)

Table 5–1: EDIT Key Commands (Cont.)

Key	Explanation
CTRL/C	In the first command, the three CTRL/C characters are part of the text object of the Insert command. EDIT treats them like any other character. In the second command string, the CTRL/C occurs at command level, which causes the editor to terminate. If no commands (other than CLOSE) are executed between the time you terminate the editor and the time you issue a REENTER command, the text buffer is preserved as it was at program termination. However, only the text buffer is preserved. The input and output files are closed, and the save and macro buffers are reinitialized. If you inadvertently terminate an editing session before the output file can be closed, you can often use the monitor CLOSE command to make permanent the portion of the output file that has already been written (see Section 4.4). You can then reenter the editor, open a new output file, and continue the editing session.
CTRL/O	Echoes as ^O and a carriage return. Inhibits printing on the terminal until completion of the current command string. Typing a second CTRL/O resumes output.
CTRL/U	Echoes as ^U and a carriage return. Deletes all characters on the current terminal input line. (Typing CTRL/U has the same effect as pressing the RUBOUT key until all the characters back to the beginning of the line are deleted.)
RUBOUT or DELETE	Deletes a character from the current command line; echoes as a backslash followed by the character deleted. Each succeeding RUBOUT you type deletes and echoes another character. An enclosing backslash prints when you type a key other than RUBOUT. This erasure is done from right to left. Since EDIT accepts multiple-line commands, RUBOUT can delete past the carriage return/line feed combination and delete characters on the previous line. You can use RUBOUT in both command and text modes.
TAB	Spaces to the next tab stop. Tab stops are positioned every eight spaces on the terminal; pressing the TAB key causes the carriage to advance to the next tab position.

CTRL/X	Echoes as ^X and a carriage return. CTRL/X causes the editor to ignore the entire command string you are currently entering. The editor prints a carriage return/line feed combination and an asterisk to indicate that you can enter another command. For example:
	
	`*IABCD`
	`EFGH^X`
	`*`
	
	A CTRL/U would cause only deletion of EFGH; CTRL/X erases the entire command.

5.4 Command Structure

EDIT commands fall into eight general categories. Table 5–2 lists these categories, the commands they include, and the sections of this manual where you will find information on the particular command.

Table 5–2: EDIT Command Categories

Category	Commands	Section
File open and close		
	Edit Backup	5.6.1.3
	Edit Read	5.6.1.1
	Edit Write	5.6.1.2
	End File	5.6.1.4
File input/output		
	EXit	5.6.2.4
	Next	5.6.2.3
	Read	5.6.2.1
	Write	5.6.2.2
Immediate mode		
	ESCAPE	5.7.2
	CTRL D	5.7.2
	CTRL G	5.7.2
	CTRL N	5.7.2
	CTRL V	5.7.2
	RUBOUT	5.7.2
Pointer location		
	Advance	5.6.3.3
	Beginning	5.6.3.1
	Jump	5.6.3.2
Search		
	Find	5.6.4.2
	Get	5.6.4.1
	Position	5.6.4.3
Text listing		
	List	5.6.5.1
	Verify	5.6.5.2
Text modification		
	Change	5.6.6.4
	Delete	5.6.6.2
	eXchange	5.6.6.5
	Insert	5.6.6.1
	Kill	5.6.6.3
Utility		
	Edit Console	5.7.1
	Edit Display	5.7.1
	Edit Lower	5.6.7.6
	Edit Upper	5.6.7.6
	Edit Version	5.6.7.5
	Execute Macro	5.6.7.4
	Macro	5.6.7.3
	Save	5.6.7.1
	Unsave	5.6.7.2

The general syntax for all the EDIT commands, except immediate mode commands, is:

[n]C[text]$

or

[n]C$

where:

n represents one of the arguments from Table 5–3

C represents a 1- or 2-letter command

text represents a string of ASCII characters

As a rule, commands are separated from one another by a single ESCAPE; however, if the command requires no text, the separating ESCAPE is not necessary. Commands are terminated by a single ESCAPE; typing a second ESCAPE begins execution. (You use ESCAPE differently when immediate mode is in effect; see Section 5.7.2.)

The syntax of display editor commands is different from the normal editing command format, and is described in Section 5.7.

5.4.1 Arguments

An argument is positioned before a command letter. It specifies either the particular portion of text to be affected by the command or the number of times to perform the command. With some commands, this specification is implicit and no argument is needed; other editing commands require an argument. Table 5–3 lists the possible arguments and their meanings.

Table 5–3: Command Arguments

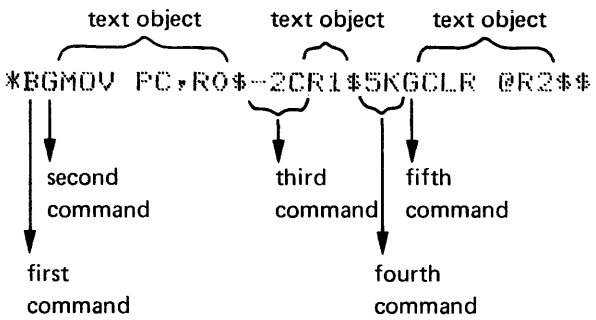
Argument	Meaning
n	Represents an integer in the range –16383 to +16383 and may, except where noted, be preceded by a plus (+) or minus (–) sign. If no sign precedes *n*, it is assumed to be a positive number. The absence of *n* implies a 1 (or –1 if a minus sign precedes a command). This argument can represent the number of characters or lines forward or backward (+ or –) to move the pointer, or it can represent the number of times to execute the operation.
0	Indicates the text between the beginning of the current line and the reference pointer (see Section 5.4.3).
/	Refers to the text between the reference pointer and the end of the text in the buffer.
=	Represents –*n*, where *n* is equal to the length of the last text argument used. Use this argument with the J, D, and C commands only.

The roles of all arguments are explained in the following sections.

5.4.2 Command Strings

All EDIT command strings are terminated by two successive ESCAPE characters. Use spaces, carriage returns, and line feeds within a command string to increase command readability. EDIT ignores them unless they appear in a text string. Commands to insert text can contain text strings that are several lines long. Each line you enter is terminated by the carriage return key, which inserts both a carriage return and a line feed character into the text. The entire command is terminated by a double ESCAPE.

You can string several commands together and execute them in sequence. For example:



where:

B	is the first command
GMOV PC,R0	is the second command (MOV PC,R0 is the text object)
-2CR1	is the third command (R1 is the text object)
5K	is the fourth command
GCLR @R2	is the fifth command (CLR @R2 is the text object)
$	separates the end of each text object from the following command
$$	executes the commands

Execution of a command string begins when you type the double ESCAPE and proceeds from left to right. EDIT ignores spaces, carriage returns, line feeds, and single ESCAPEs, except when they are part of a text string. Thus, example (1) has the same result as example (2):

(1)
```
*BGMOV R0$=CCLR R1$AV$$
```

(2)
```
*B$ GMOV R0$
=CCLR R1$
A$ V$$
```

5.4.3 Current Location Pointer

Most EDIT commands function with respect to a movable reference pointer that is normally located between the most recent character operated upon and the next character in the buffer. It is important to think of this pointer as being between two characters, and never directly on a character. At the start of editing operations, the pointer precedes the first character in the buffer, although it is not displayed on the console terminal. At any time during the editing procedure, think of the pointer as representing the current position of the editor in the text. The pointer moves during editing operations according to the type of editing operation being performed. Refer to text in the buffer as so many characters or lines preceding or following the pointer.

5.4.4 Character- and Line-Oriented Command Properties

Edit commands are either character-oriented or line-oriented: character-oriented commands affect a specified number of characters preceding or following the pointer; line-oriented commands operate on entire lines of text.

The argument of character-oriented commands specifies the number of characters in the buffer on which to operate. If *n* is unsigned (positive), the command moves the pointer in a forward direction. If *n* is preceded by a minus sign (negative), the command moves the reference pointer backward. LF, RET, and null characters, although not printed, are embedded in text lines, counted as characters in character-oriented commands, and treated as any other text characters. When you press the RET key, both a carriage return and a line feed character are inserted into the text. For example, assume the pointer is positioned as indicated in the following text (↑ represents the current position of the pointer):

```
MOV    #VECT,R2 (RET) (LF)
                      ↑
CLR    @R2 (RET) (LF)
```

The EDIT command -2J moves the pointer back two characters to precede the carriage return character.

```
MOV    #VECT, R2↑(RET) (LF)
CLR    @R2 (RET) (LF)
```

The command 10J advances the pointer forward ten characters and places it between the RET and LF characters at the end of the second line. Note that the tab character preceding @R2 is also counted as a single character.

```
MOV    #VECT,R2 (RET) (LF)
CLR    @R2 (RET)↑(LF)
```

Finally, to place the pointer after the C in the first line, use a -14J command. The J (Jump) command is explained in Section 5.6.3.2.

```
MOV    #VECT↑,R2 (RET) (LF)
CLR    @R2 (RET) (LF)
```

When you use line-oriented commands, the command argument specifies the number of lines on which to operate. Because EDIT counts the line-terminating characters to determine the number of lines on which to operate, an argument *n* does not affect the same number of lines forward (positive) as it affects backward (negative). For example, the argument -1 applies to the line beginning with the first character following the second previous end-of-line and ending with the character preceding the pointer. The argument 1 in a line-oriented command, however, applies to the text beginning with the first character following the pointer and ending at the first end-of-line. Thus, if the pointer is at the center of the line, the argument -1 affects one and one-half lines backward from the pointer and the argument 1 affects one-half line beyond the pointer.

For example, assume the buffer contains:

```
MOV   PC,R1(RET)(LF)
ADD   ↑#DRIV-.,R1(RET)(LF)
MOV   #VECT,R2(RET)(LF)
CLR   @R2(RET)(LF)
```

The command to advance the pointer one line (1A) causes the following change:

```
MOV   PC,R1(RET)(LF)
↑ADD  #DRIV-.,R1(RET)(LF)
↑MOV  #VECT,R2(RET)(LF)
CLR   @R2(RET)(LF)
```

The command 2A moves the pointer over two RET LF combinations to precede the fourth line:

```
MOV   PC,R1(RET)(LF)
ADD   #DRIV-.,R1(RET)(LF)
MOV   #VECT,R2(RET)(LF)
↑CLR  @R2 (RET)(LF)
↑
```

Assume the buffer contains:

```
MOV   PC,R1(RET)(LF)
ADD   #DRIV-.,R1(RET)(LF)
MOV   #VECT,R2(RET)(LF)
CLR   @R2 ↑(RET)(LF)
```

A command of -1A moves the pointer back by one and one-half lines to precede the second line.

```
MOV   PC,R1(RET)(LF)
↑ADD  #DRIV-.,R1(RET)(LF)
↑MOV  #VECT,R2(RET)(LF)
CLR   @R2(RET)(LF)
```

Now a command of -1A moves the pointer back by only one line.

```
MOV    PC,R1(RET)(LF)
ADD    #DRIV-.,R1(RET)(LF)
MOV    #VECT,R2(RET)(LF)
CLR    @R2(RET)(LF)
```

5.4.5 Command Repetition

You can execute portions of a command string more than once by enclosing the portion in angle brackets (<>) and preceding the left angle bracket with the number of iterations you desire. The syntax is:

n<command>

For example:

```
C1$C2$n<C3$C4$>C5$$
```

where:

C represents a command

n represents an iteration argument

Commands C1 and C2 each execute once, then commands C3 and C4 execute *n* times. Finally, command C5 executes once and the command line is finished. The iteration argument (*n*) must be a positive number (in the range 1 through 16383); if unspecified, it is assumed to be 1. If the number is negative or too large, an error message prints. You can nest iteration brackets up to 20 levels. Before execution, EDIT checks command lines to make certain the brackets are correctly used and match.

Enclosing a portion of a command string in iteration brackets and preceding it with an iteration argument (*n*) has the same result as typing that portion of the string *n* times. Thus, example (1) and example (2) are equivalent.

```
(1)  *BGAAA$3<-DIB$-J>V$$
(2)  *BGAAA$-DIB$-J-DIB$-J-DIB$-JV$$
```

Similarly, the following two strings are equivalent:

```
*B3<2<AD>V>$$
*BADADVADADVADADV$$
```

The following bracket structures are examples of legal usage:

```
<<><<<><>>>>
<<<>>><><>
```

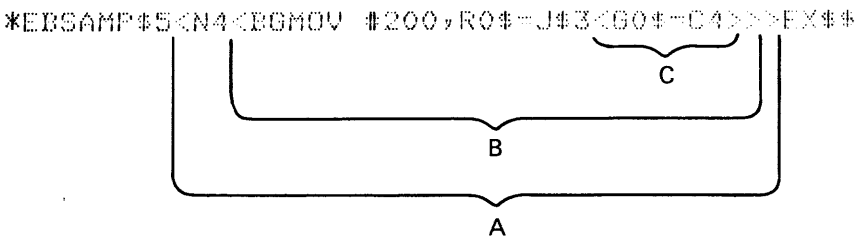
The following bracket structures are examples of combinations that will cause an error message:

```
><><
<<<>>
```

During command repetition, execution proceeds from left to right until a right bracket is encountered. EDIT then returns to the last left bracket encountered, decreases the iteration counter, and executes the commands within the brackets. When the counter is decreased to 0, EDIT looks for the next iteration count to the left and repeats the same procedures. The overall effect is that EDIT works its way to the innermost brackets and then works its way back again. The most common use for iteration brackets is found in commands, such as Unsave (U), that do not accept repeat counts. For example:

```
*3<U>$$
```

Assume you want to read a file called SAMP (stored on device DK:), and you want to change the first four occurrences of the instruction MOV #200,R0 on each of the first five pages to MOV #244,R4. Enter the following command line:

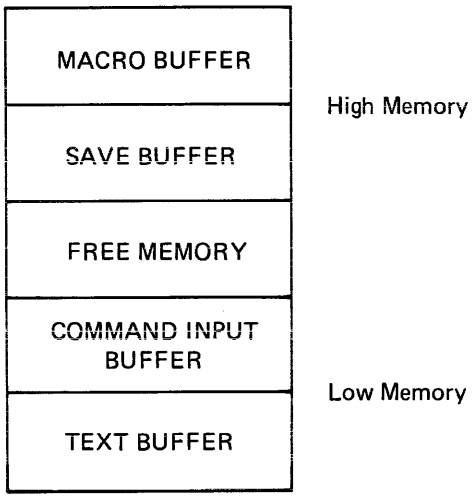


The command line contains three sets of iteration loops (A,B,C) and executes as follows:

Execution initially proceeds from left to right; EDIT opens the file SAMP for input and reads the first page into memory. EDIT moves the pointer to the beginning of the buffer and initiates a search for the character string MOV #200,R0. When it finds the string, EDIT positions the pointer at the end of the string, but the =J command moves the pointer back, so that it is positioned immediately preceding the string. At this point, execution has passed through each of the first two sets of iteration loops (A,B) once. The innermost loop (C) is next executed three times, changing the 0s to 4s. Control now moves back to pick up the second iteration of loop B, and again moves from left to right. When loop C has executed three times, control again moves back to loop B. When loop B has executed a total of four times, control moves back to the second iteration of loop A, and so forth, until all iterations have been satisfied.

5.5 Memory Usage

The memory area used by the editor is divided into four logical buffers as follows:



The text buffer contains the current page of text you are editing, and the command input buffer holds the command you are currently typing at the terminal. If a command you are currently entering is within ten characters of exceeding the space available in the command buffer, the following message prints on the terminal.

```
?EDIT-W-Command buffer almost full
```

If you can complete the command within ten characters, you can finish entering the command; otherwise you should press the ESCAPE key twice to execute that portion of the command line already completed. The warning message prints each time you enter a character in one of the last ten spaces.

If you attempt to enter more than ten characters, EDIT prints the following message and aborts the command.

```
?EDIT-F-Command buffer full;no command(s) executed
```

The save buffer contains text stored with the Save (S) command, and the macro buffer contains the command string macro entered with the Macro (M) command. (Both commands are explained in Section 5.6.7.)

EDIT does not allocate space for the macro and save buffers until an M or S command executes. Once you enter an M or S command, a 0M or 0U (Unsave) command executes to return that space to the free area.

The size of each buffer automatically expands and contracts to accommodate the text you are entering; if there is not enough space available to accommodate required expansion of any of the buffers, EDIT prints the error message:

```
?EDIT-F-Insufficient memory
```

5.6 Editing Commands

This section describes the commands and procedures required to:

- Read text from the input files to the buffer
- Create a backup version of the file
- List the contents of the buffer on the terminal
- Move the reference pointer
- Locate characters or strings of characters within the text buffer
- Insert, relocate, or delete text in the buffer
- Close the output file
- Terminate the editing session

The following sections are arranged, in order, by category of command function, as illustrated in Table 5–2.

5.6.1 File Open and Close Commands

You can use file open and close commands to:

- Open an existing file for input and prepare it for editing (ER)
- Open a file for output of newly created or edited text (EW)
- Open an existing file for editing and create a backup version of it (EB)
- Close an open output file (EF)

5.6.1.1 Edit Read – The Edit Read (ER) command opens an existing file for input and prepares it for editing. Only one file can be open for input at a time.

The syntax of the command is:

ERdev:filnam.typ$

The argument *dev:filnam.typ* is limited to 19 characters and specifies the file to be opened. If you do not specify a device, DK: is assumed. If a file is currently open for input, EDIT closes that file and opens the new one.

Edit Read does not input a page of text nor does it affect the contents of the other user buffers.

With Edit Read you can close a file that is already open for input and reposition EDIT at the beginning of the file. The first Read command following any Edit Read command inputs the first page of the file.

This command string opens the file SAMP.MAC on device DT1:.

```
*ERDT1:SAMP.MAC$$
```

NOTE

If you enter EDIT with the monitor EDIT/INSPECT or EDIT/OUTPUT command, an Edit Read command is automatically performed on the file named in the EDIT command.

5.6.1.2 Edit Write – The Edit Write (EW) command opens a file for output of newly created or edited text. However, no text is output and the contents of the buffers are not affected. Only one file can be open for output at a time. EDIT closes any output files currently open and preserves any edits made to the file.

The syntax of the command is:

EWdev:filnam.typ[n]$

The argument *dev:filnam.typ[n]* is limited to 19 characters and is the name you assign to the output file being opened. If you do not specify a device, DK: is assumed. The optional argument [n] is a decimal number that represents the length of the file to be opened. Note that the square brackets ([]) are part of this argument and must be typed. If you do not specify [n], the system will default to either the larger of one-half the largest available space, or the second largest available space. If this is not adequate for the output file size, you must close this file and open another when this one becomes full. You should use the [n] construction whenever there is doubt as to whether enough space is available on the device for one output file.

If a file with the same name already exists on the device, EDIT deletes the existing file when you type an Exit, End File, or another Edit Write command. EDIT prints the warning message:

```
?EDIT-W-Supersedins existins file
```

The following command, for example, opens FILE.BAS on device DK: and allocates 11 blocks of space for it.

```
*EWFILE.BAS[11]$$
```

NOTE

If you enter EDIT with the monitor EDIT/CREATE command, an Edit Write command is automatically performed on the file named in the EDIT command. If you enter EDIT with the monitor EDIT/OUTPUT command, an Edit Write is automatically performed on the file named with the /OUTPUT option.

5.6.1.3 Edit Backup – The Edit Backup (EB) command opens an existing file for editing and at the same time creates a backup version of the file. EDIT closes any input and output file currently open. No text is read or written with this command.

The syntax of the command is:

EBdev:filnam.typ[n]$

The argument *dev:filnam.typ[n]* is limited to 19 characters. If you do not specify a device, DK: is assumed. The argument [n] is optional and represents the length of the file to be opened; if you do not specify [n], the system defaults to either the larger of one-half the largest available space, or the second largest available space.

The file you indicate in the command line must already exist on the device you designate, because text will be read from this file as input. At the same time, EDIT opens an output file under the same file name and file type. When the output file is closed, EDIT renames the original file (used as input) with the current file name and a .BAK file type and deletes any previous file with this file name and a .BAK file type. EDIT closes the new output file and assigns it the name you specify in the EB command. This renaming of files takes place when an Exit, End File, or subsequent Edit Write or Edit Backup command executes. If you terminate the editing session with a CTRL/C command before the output file is closed, the new output file is not made permanent, and the renaming of the current version to .BAK does not take place. Thus:

```
*EBSY:BAS1.MAC$$
```

This command opens BAS1.MAC on device SY:. When editing is complete, the old BAS1.MAC becomes BAS1.BAK, and the new file becomes BAS1.MAC. EDIT deletes any previous version of BAS1.BAK.

NOTE

In EB, ER, and EW commands, leading spaces between the command and the file name are not permitted because EDIT assumes the file name to be a text string. All dev:file.typ specifications for EB, ER, and EW commands conform to RT–11 conventions for file naming. File names entered in command strings used with other system programs have identical specifications.

If you enter EDIT with the monitor EDIT command, an Edit Backup command is automatically performed on the file named in the EDIT command.

5.6.1.4 End File – The End File (EF) command closes the current output file and makes it permanent. You can use the EF command to create an output file from a section of a large input file or to close an output file that is full before you open another file. Modifiers are illegal with an EF command. Note that an implied EF command is included in EW and EB commands.

The syntax of the command is:

EF

Table 5–4 illustrates the relationship between the file open and close commands and the buffers and files themselves.

Table 5–4: EDIT Commands and File Status

Command	File Status		
	Input	Text Buffer	Output
ERXXX$	Opens *XXX* for input; closes existing input file, if any	Unchanged	Unchanged
EWXXX$	Unchanged	Unchanged	Opens *XXX* for output; closes existing output file, if any; performs .BAK renaming if EB is in effect
EBXXX$	Opens *XXX* for input; closes existing input file, if any	Unchanged	Opens a temporary file for output; closes existing output file, if any; performs .BAK renaming if EB is in effect
EF$	Unchanged	Unchanged	Closes output file; performs .BAK renaming if EB is in effect
EX$	Copies to output file	Copies to output file	Closes output file after copying complete; performs .BAK renaming if EB is in effect

5.6.2 File Input/Output Commands

You use file input/output commands to:

- Read text from an input file into the buffer
- Copy lines of text from the buffer into an output file
- Terminate the editing session

5.6.2.1 Read – Before you can edit text, you must read the input file into the buffer. The Read (R) command reads a page of text from the input file (previously specified in an ER or EB command) and appends it to the current contents of the text buffer (contents can be empty).

The command is:

R

No arguments are used with the R command. If the buffer contains text when the R command is executed, the pointer does not move; however, if the buffer does not contain text, the pointer is placed at the beginning of the buffer. EDIT transfers text to the buffer until one of the following conditions occurs:

1. A form feed character, signifying the end of the page, is encountered.
2. The text buffer is 500 characters from being full. (When this condition occurs, the Read command inputs up to the next carriage return/line

feed combination, then returns to command mode. An asterisk prints as though the read were complete, but text will not have been fully input.)

3. An end-of-file is encountered. (The *?EDIT-F-End of input file* message prints when all text in the file has been read into memory and no more input is available.)

The maximum number of characters you can bring into memory with an R command depends on the system configuration and the memory requirements of other system components. EDIT prints an error message if the read exceeds the memory available or if no input is available.

The following example creates a file using the EB and R commands.

```
*EBSJK1.BAS$$
```

This command opens SJK1.BAS on DK: and permits modification.

```
*R/L$$
THIS IS PAGE ONE OF
FILE SJK1.BAS.
```

This command reads the first page of SJK1.BAS into the buffer. The pointer is placed at the beginning of the buffer. /L lists the contents of the buffer on the terminal, beginning at the pointer and ending with the last character in the buffer.

5.6.2.2 Write – The Write (nW) command copies lines of text from the text buffer to the output file (as specified in the EW or EB command). The contents of the buffer are not altered and the pointer is left unchanged (unless an output error occurs).

NOTE

EDIT uses a system of intermediate buffers to store output before it writes the data to an output file. The Write command logically writes to the file, but output to a device does not occur until the intermediate buffer fills. When the editor closes a file (that is, after you issue an EF, EB, EX, or EW command), text is written from the buffer to the file and the file is complete. If the editor does not close a file (if you exit with CTRL/C and use the CLOSE command), it is possible that the output file will be missing the last 512 characters.

The syntax of the command is:

nW

The argument you supply with the W command determines the lines of text to copy. Table 5–5 lists the arguments for the W command.

Table 5–5: Write Command Arguments

Argument	Function
n	Writes *n* lines of text beginning at the pointer and ending with the *n*th end-of-line character to the output file.
–n	Writes *n* lines of text to the output file beginning with the first character on the *–n*th line and terminating at the pointer.
0	Writes to the output file the current line up to the pointer.
	Writes to the output file the text between the pointer and the end of the buffer.

If the buffer is empty when the write executes, no characters are output.

The following examples illustrate the use of the W command.

(1) `*5W$$`

In example (1), the command writes the five lines of text following the pointer into the current output file.

(2) `*-2W$$`

In example (2), the command writes the two lines of text preceding the pointer into the current output file.

(3) `B/W$$`

In example (3), the command writes the entire text buffer to the current output file.

NOTE

If an output file fills while a Write command is executing, EDIT prints the *?EDIT-F-Output file full* message. In this case, EDIT positions the reference pointer after the last character it wrote successfully. You can then use the following recovery procedure:

1. Close the current output file (EF command).
2. Open a new output file (EW command).
3. Delete the characters just written by using -nD or -nK, where *n* is any arbitrary number that exceeds the number of lines or characters in the buffer.
4. Resume output.

5.6.2.3 Next – The Next (nN) command writes the contents of the text buffer to the output file, deletes the text from the buffer, and reads the next page of the input file into the buffer. The pointer is positioned at the beginning of the buffer. The syntax of the command is:

nN

If you specify the argument *n* with the Next command, the sequence is executed *n* times.

If EDIT encounters the end of the input file when trying to execute an N command, it prints *?EDIT-F-End of input file* to indicate that no further text remains in the input file. Since the contents of the buffer have already been transferred to the output file, the buffer is empty.

Using the N command is a quick way to write edited text to the output file and set up the next page of text in the buffer. The N command functions as though it were a combination of the Write, Delete, Read, and Beginning commands. (Delete is a text modification command, described in Section 5.6.6.2; the Beginning command is a pointer relocation command, described in Section 5.6.3.1.) Using the N command with an argument is a convenient way to set up text in the buffer, if you already know its page location. The N command operates in a forward direction only; therefore, you cannot specify negative arguments with an N command.

In the following example, an N command copies an input file with more than one page of text to the output file.

```
*EBDK:TEST.MAC$$
```

This command opens the file TEST.MAC on device DK: and creates a new file entitled TEST.MAC for output.

```
*N/L$$
THIS IS PAGE ONE OF
FILE TEST.MAC.
```

This command reads the first page of the input file, TEST.MAC, into the buffer and lists the entire page on the terminal.

```
*N/L$$
?EDIT-F-End of input file
*
```

This command transfers the contents of the buffer to the output file, clears the buffer, and encounters the end of the file. Because it cannot complete the N sequence, EDIT prints *?EDIT-F-End of input file* on the terminal. The buffer is empty and the entire input file has been written to the output file.

5.6.2.4 EXit – Type the Exit (EX) command to terminate an editing session. The Exit command:

- Writes the text buffer to the output file
- Transfers the remainder of the input file to the output file
- Closes all open files
- Renames the backup file with a .BAK file type if an EB command is in effect
- Returns control to the monitor

The command is:

EX

No arguments are accepted. Essentially, Exit copies the remainder of the input file into the output file and returns to the monitor. Exit is legal only when there is an output file open. If an output file is not open and you want to terminate the editing session, return to the monitor with CTRL/C.

NOTE

> You must issue an EF or EX command in order to make an output file permanent. If you use CTRL/C to return to the monitor without issuing an EF command, the current output file will not be saved. (You can, however, make permanent that portion of the text file that has already been written out, by using the monitor CLOSE command.)

An example of the contrasting uses of the EF and EX commands follows. Assume an input file, SAMPLE, contains several pages of text. The first and second pages of the file will be made into separate files called SAM1 and SAM2, respectively; the remaining pages of text will then make up the file SAMPLE. This can be done by using these commands:

```
*EWSAM1$$
*ERSAMPLE$$
*RNEF$$
*EWSAM2$$
*NEF$$
*EWSAMPLE$EX$$
```

Note that the EF commands are not necessary in this example since the EW command closes a currently open output file before opening another.

5.6.3 Pointer Relocation Commands

Pointer relocation commands allow you to change the current location of the reference pointer within the text buffer.

5.6.3.1 Beginning – The Beginning (B) command moves the current location of the pointer to the beginning of the text buffer.

The command is:

B

There are no arguments.

For example, assume the buffer contains:

```
MOVB  5(R1),@R2
ADD   R1,(R2)+
CLR   @R2
MOVB  6(R1),@R2
```

The B command moves the pointer to the beginning of the text buffer.

```
*B$$
```

The text buffer now looks like this:

```
↑MOVB  5(R1),@R2
 ADD   R1,(R2)+
 CLR   @R2
 MOVB  6(R1),@R2
```

5.6.3.2 Jump – The Jump (nJ) command moves the pointer past a specified number of characters in the text buffer. The syntax of the command is:

nJ

Table 5–6 shows the arguments for the J command.

Table 5–6: Jump Command Arguments

Argument	Meaning
(+ or –) n	Moves the pointer (forward or backward) *n* characters.
0	Moves the pointer to the beginning of the current line (equivalent to 0A).
/	Moves the pointer to the end of the text buffer (equivalent to /A).
=	Moves the pointer backward *n* characters, where *n* equals the length of the last text argument used.

Negative arguments move the pointer toward the beginning of the buffer; positive arguments move it toward the end. Jump treats carriage returns, line feeds, and form feed characters the same as any other character, counting one buffer position for each one.

The following examples illustrate the J command.

```
*3J$$
```

This command moves the pointer ahead three characters.

```
*-4J$$
```

This command moves the pointer back four characters.

```
*B$GABC$=J$$
```

This command moves the pointer so that it immediately precedes the first occurrence of ABC in the buffer.

5.6.3.3 Advance – The Advance (nA) command is similar to the Jump command except that it moves the pointer a specific number of lines (rather than single characters) and leaves it positioned at the beginning of the line. The syntax of the command is:

nA

Table 5–7 lists the arguments for the A command and their meanings.

Table 5–7: Advance Command Arguments

Argument	Meaning
n	Moves the pointer forward *n* lines and positions it at the beginning of the *n*th line.
–n	Moves the pointer backward past *n* carriage return/line feed combinations and positions it at the beginning of the – *n*th line.
0	Moves the pointer to the beginning of the current line (equivalent to 0J).
/	Moves the pointer to the end of the text buffer (equivalent to /J).

Following are examples that use the A command.

```
*3A$$
```

This command moves the pointer ahead three lines.

Assume the buffer contains:

```
CLR     @R2
        ↑
```

The following command moves the pointer to the beginning of the current line:

```
*0A$$
```

Now the buffer looks like this:

```
↑CLR    @R2
```

5.6.4 Search Commands

Use search commands to locate characters or strings of characters within the text buffer.

NOTE

Search commands always have positive arguments. They search ahead in the file. This means that to search for a character string that has already been written to the output file, you must first close the currently open file (with EX) and then edit the file that was just used for output (with EB).

5.6.4.1 Get – The Get (nG) command is the basic search command in EDIT. It searches the current text buffer for the nth occurrence of a specific text string, starting at the current location of the pointer. If you do not supply the argument *n*, EDIT searches for the first occurrence of the text object. The search terminates when EDIT either finds the *n*th occurrence or encounters

the end of the buffer. If the search is successful, EDIT positions the pointer to follow the last character of the text object. EDIT notifies you of an unsuccessful search by printing *?EDIT-F-Search failed.* In this instance, EDIT positions the pointer after the last character in the buffer.

The syntax of the command is:

nGtext$

The argument *n* must be positive. If you omit it, EDIT assumes it to be 1.

The text string may be any length and must immediately follow the G command. EDIT makes the search on the portion of the text between the pointer and the end of the buffer.

For example, assume the pointer is at the beginning of the buffer shown below.

```
↑MOV   PC,R1
 ADD   #DRIV-.,R1
 MOV   #VECT,R2
 CLR   @R2
 MOVB  5(R1),@R2
 ADD   R1,(R2)+
 CLR   @R2
 MOVB  6(R1),@R2
```

The following command searches for the first occurrence of the characters ADD following the pointer and places the pointer after the searched characters.

```
*GADD$$
```

Now the buffer looks like this:

```
MOV   PC,R1
ADD↑  #DRIV-.,R1
```

The next command searches for the third occurrence of the characters @R2 following the pointer and leaves the pointer immediately following the text object.

```
*3G@R2$$
```

The buffer is changed to:

```
ADD   R1,(R2)+
CLR   @R2↑
```

After successfully completing a search command, EDIT positions the pointer immediately following the text object. Using a search command in combination with =J places the pointer in front of the text object, as follows:

```
*GTEST$=J$$
```

This command combination places the pointer before TEST in the text buffer.

5.6.4.2 Find – The Find (nF) command starts at the current pointer location and searches the entire input file for the *n*th occurrence of the text string. If EDIT does not find the *n*th occurrence of the text string in the current buffer, it automatically performs a Next command and continues the search on the new text in the buffer. When the search is successful, EDIT leaves the pointer immediately following the *n*th occurrence of the text string. If the search fails (that is, EDIT detects the end-of-file for the input file and does not find the *n*th occurrence of the text string), EDIT prints *?EDIT-F-Search failed.* In this instance, EDIT positions the pointer at the beginning of an empty text buffer. When you use the F command, EDIT deletes the contents of the buffer after writing it to the output file.

The syntax of the command is:

nFtext$

The argument *n* must be positive. EDIT assumes it to be 1 if you do not supply another value.

You can use an F command to copy all remaining text from the input file to the output file by specifying a nonexistent text object. The Find command functions like a combination of the Get and Next commands.

The following example uses the F command.

```
*2FMOVB         6(R1),@R2$$
```

This command searches the entire input file for the second occurrence of the text string MOVB 6(R1),@R2. EDIT places the pointer following the text string. EDIT writes the contents of each unsuccessfully searched buffer to the output file.

5.6.4.3 Position – The Position (nP) command is identical to the Find (F) command with one exception. The F command transfers the contents of the text buffer to the output file as each page is unsuccessfully searched, but the P command deletes the contents of the buffer after it is searched without writing any text to the output file.

The syntax of the command is:

nPtext$

The argument *n* must be positive. If you omit it, EDIT assumes it to be 1.

The nP command searches each page of the input file for the *n*th occurrence of the text object starting at the pointer and ending with the last character in the buffer. If EDIT finds the *n*th occurrence, it positions the pointer following the text object, deletes all pages preceding the one containing the text object, and positions the page containing the text object in the buffer.

If the search is unsuccessful, EDIT clears the buffer but does not transfer any text to the output file. EDIT positions the pointer at the beginning of an empty text buffer.

The Position command is a combination of the Get, Delete, and Read commands; it is most useful as a means of placing the pointer in the input file. For example, if your aim in the editing session is to create a new file from the second half of the input file, a Position search saves time.

The following example uses the P command.

```
*P3$$
```

This command searches the input file for the first occurrence of the text object, 3. EDIT positions the pointer after the text object.

```
*0L$$
INPUT FILE PAGE 3
```

This command lists on the terminal the current line up to the pointer.

5.6.5 Text Listing Commands

5.6.5.1 List – The List (nL) command prints at the terminal lines of text as they appear in the buffer. The syntax of the command is:

nL

An argument preceding the L command indicates the portion of text to print. For example, the command, 2L, prints on the terminal the text beginning at the pointer and ending with the second end-of-line character. The pointer is not altered by the L command. Table 5–8 lists arguments and their effect upon the list command.

Table 5–8: List Command Arguments

Argument	Meaning
n	Prints at the terminal *n* lines beginning at the pointer and ending with the *n*th end-of-line character.
–n	Prints all characters beginning with the first character on the –*n*th line and terminating at the pointer.
0	Prints the current line up to the pointer. Use this command to locate the pointer within a line.
/	Prints the text between the pointer and the end of the buffer.

These examples illustrate the use of the L command.

```
*-2L$$
```

This command prints all characters starting at the second preceding line and ending at the pointer.

```
*4L$$
```

This line prints all characters beginning at the pointer and terminating at the fourth carriage return/line feed combination.

Assuming the pointer location is:

```
MOVB  5(R1),@R2
ADD   R1,(R2)+
  ↑
```

The following command prints the previous one and one-half lines up to the pointer:

```
*-1L$$
```

The terminal output now looks like this:

```
MOVB  5(R1),@R2
ADD
```

5.6.5.2 Verify – The Verify (V) command prints at the terminal the entire line in which the pointer is located. It provides a ready means of determining the location of the pointer after a search completes and before you give any editing commands. (The V command combines the two commands 0LL.) You can also type the V command after an editing command to allow proofreading of the results. No arguments are allowed with the V command. The location of the pointer does not change.

5.6.6 Text Modification Commands

You can use the following commands to insert, relocate, and delete text in the text buffer.

5.6.6.1 Insert – The Insert (I) command is the basic command for inserting text. EDIT inserts the text you supply at the location of the pointer and then places the pointer after the last character of the new text.

The syntax of the command is:

Itext$

No arguments are allowed with the insert command. The text string is limited only by the size of the text buffer and the space available. All characters are legal, except ESCAPE, which terminates the text string.

NOTE

If you forget to type the I command, the editor will interpret the text as commands.

EDIT automatically protects the text buffer from overflowing during an insert. If the I command is the first command in a multiple command line, EDIT ensures that there will be enough space for the insert to be executed at least once. If repetition of the command exceeds the available memory, an error message prints.

The following example illustrates the I command.

```
*IMOV           #BUFF,R2
MOV             #LINE,R1
MOVB            -1(R2),R0$$
*
```

This command inserts the text at the current location of the pointer and leaves the pointer positioned after R0.

DIGITAL recommends that you insert large amounts of text into the file in small sections rather than all at once. This way, you are less vulnerable to loss of time and effort due to machine failure or human error. This is the recommended procedure for inserting large amounts of text:

1. Open the file with the EB command.
2. Insert or edit a few pages of text.
3. Insert a unique text string (like ????) to mark your place.
4. Use the Exit command to preserve the work you have done so far.
5. Start again, using the F command to search for the unique string you used to mark your place.
6. Delete your marker and continue editing.

5.6.6.2 Delete – The Delete (nD) command is a character-oriented command that deletes *n* characters in the text buffer, beginning at the current location of the pointer. The syntax of the command is:

nD

If you do not specify *n*, EDIT deletes the character immediately following the pointer. Upon completion of the D command, EDIT positions the pointer immediately before the first character following the deleted text. Table 5–9 lists arguments for the D command.

Table 5–9: Delete Command Arguments

Argument	Meaning
n	Deletes *n* characters following the pointer. Places the pointer before the first character following the deleted text.
–n	Deletes *n* characters preceding the pointer. Places the pointer before the first character following the deleted text.
0	Deletes the current line up to the pointer. The position of the pointer does not change (equivalent to 0K).
/	Deletes the text between the pointer and the end of the buffer. Positions the pointer at the end of the buffer (equivalent to /K).
=	Deletes –*n* characters, where *n* equals the length of the last text argument used.

The following examples illustrate the use of the D command.

```
*-2D$$
```

This command deletes the two characters immediately preceding the pointer.

```
*B$FMOV R1$=D$$
```

This command string deletes the text string MOV R1 (=D in combination with a search command deletes the indicated text string).

Assume the text buffer contains the following:

```
ADD    R1,(R2)+
CLR    @R2
       ↑
```

The following command deletes the current line up to the pointer:

```
*0D$$
```

The buffer now contains:

```
ADD    R1,(R2)+
@R2
↑
```

5.6.6.3 Kill – The Kill (nK) command removes *n* lines of text (including the carriage return and line feed characters) from the page buffer, beginning at the pointer and ending with the *n*th end-of-line. The syntax of the command is:

nK

EDIT places the pointer at the beginning of the line following the deleted text. Table 5–10 describes each argument and its effect upon the Kill command.

Table 5–10: Kill Command Arguments

Argument	Meaning
n	Removes the character string (including the carriage return/line feed combination) beginning at the pointer and ending at the *n*th end-of-line.
–n	Removes the character string beginning at the *n*th end-of-line preceding the pointer and ending at the pointer. Thus, if the pointer is at the center of a line, the modifier –1 deletes one and one-half lines preceding it.
0	Removes the current line up to the pointer (equivalent to 0D).
/	Removes the characters beginning at the pointer and ending with the last line in the text buffer (equivalent to /D).

The following examples illustrate the K command.

```
*2K$$
```

This command deletes lines starting at the current location of the pointer and ending at the second carriage return/line feed combination.

Assume the text buffer contains the following:

```
ADD   R1,(R2)+
CLR   @R2
MOVB  6(R1),@R2
```

This command removes the characters beginning at the pointer and ending with the last line in the text buffer:

```
*/K$$
```

The buffer now contains:

```
ADD   R1,(R2)+
CLR
```

Kill and Delete commands perform the same function, except that Kill is line-oriented and Delete is character-oriented.

5.6.6.4 Change – The Change (nC) command changes a specific number of characters following the pointer. The syntax of the command is:

nCtext

A C command is equivalent to a Delete command followed by an Insert command. You must insert a text object following the nC command. Table 5–11 lists each argument and its effect upon the C command.

Table 5–11: Change Command Arguments

Argument	Meaning
n	Replaces *n* characters following the pointer with the specified text. Places the pointer after the inserted text.
–n	Replaces *n* characters preceding the pointer with the specified text. Places the pointer after the inserted text.
0	Replaces the current line up to the pointer with the specified text. Places the pointer after the inserted text (equivalent to 0X).
/	Replaces the text beginning at the pointer and ending with the last character in the buffer. Places the pointer after the inserted text (equivalent to /X).
=	Replaces –*n* characters with the indicated text string, where *n* represents the length of the last text argument used.

The size of the text is limited only by the size of the text buffer and the space available. All characters are legal except ESCAPE, which terminates the text string.

If the C command is to be executed more than once (that is, it is enclosed in angle brackets) and if there is enough space available for the command to be entered, it will be executed at least once (provided it appears first in the command string). If repetition of the command exceeds the available memory, an error message prints.

The following examples illustrate the C command.

```
*5C#VECT$$
```

This command replaces the five characters to the right of the pointer with #VECT.

Assume the text buffer contains the following:

```
CLR    @R2
MOV    5(R1),@R2
```

The next command replaces the current line up to the pointer with the specified text.

```
*0CADDB$$
```

The buffer now contains:

```
CLR    @R2
ADDB   5(R1),@R2
```

You can use =C with a Get command to replace a specific text string. Here is an example:

```
*GFIFTY:$=CFIVE:$
```

This command finds the text string FIFTY: and replaces it with FIVE:.

5.6.6.5 Exchange – The Exchange (nX) command is similar to the change command except that it changes lines of text, instead of a specific number of characters. The syntax of the command is:

nXtext

The nX command is identical to an nK command followed by an Insert command. Table 5–12 lists the Exchange command arguments.

Table 5–12: Exchange Command Arguments

Argument	Meaning
n	Replaces *n* lines, including the carriage return and line feed characters following the pointer. Places the pointer after the inserted text.
–n	Replaces *n* lines, including the carriage return and line feed characters preceding the pointer. Positions the pointer after the inserted text.
0	Replaces the current line up to the pointer with the specified text. Positions the pointer after the inserted text (equivalent to 0C).
/	Replaces the text beginning at the pointer and ending with the last character in the buffer with the specified text (equivalent to /C). Positions the pointer after the inserted text.

All characters are legal in the text string except ESCAPE, which terminates the text.

If the X command is enclosed within angle brackets to allow more than one execution, and if there is enough memory space available for the X command to be entered, EDIT executes it at least once (provided it is first in the command string). If repetition of the command exceeds the available memory, an error message prints.

The following example illustrates the X command.

```
*2XADD  R1,(R2)+
CLR     @R2
$$
*
```

This command exchanges the two lines to the right of the pointer with the text string.

5.6.7 Utility Commands

During the editing session, you can store text in external buffers and subsequently restore this text when you need it later in the editing session. The following sections describe the commands that perform this function.

5.6.7.1 Save – The Save (nS) command lets you store text in an external buffer called a save buffer (described previously in Section 5.5), and subsequently insert it in several places in the text.

The syntax of the command is:

nS

The Save command copies *n* lines, beginning at the pointer, into the save buffer. The S command operates only in the forward direction; therefore, you cannot use a negative argument. The Save command destroys any previous contents of the save buffer; however, EDIT does not change the location of the pointer or the contents of the text buffer.

If you specify more characters than the save buffer can hold, EDIT prints *?EDIT-F-Insufficient memory*. None of the specified text is saved.

For example, assume the text buffer contains the following assembly language subroutine:

```
;SUBROUTINE MSGTYP
;WHEN CALLED, EXPECTS R0 TO POINT TO AN
;ASCII MESSAGE THAT ENDS IN A ZERO BYTE,
;TYPES THAT MESSAGE ON THE USER TERMINAL

MSGTYP:  TSTB   (R0)           ;DONE?
         BEQ    MDONE          ;YES-RETURN
MLOOP:   TSTB   @#177564       ;NO-IS TERMINAL READY?
         BPL    MLOOP          ;NO-WAIT
         MOVB   (R0)+,@#177566 ;YES PRINT CHARACTER
         BR     MSGTYP         ;LOOP
MDONE:   RTS    PC             ;RETURN
```

The following command stores the entire subroutine in the save buffer (assuming the pointer is at the beginning of the buffer):

```
*12S$$
```

You can insert the contents of the save buffer into a program whenever you choose by using the Unsave command.

5.6.7.2 Unsave – The Unsave (U) command inserts the entire contents of the save buffer into the text buffer at the pointer and leaves the pointer positioned following the inserted text. You can use the U command to move blocks of text or to insert the same block of text in several places. Table 5–13 lists the U commands and their meanings.

Table 5–13: Unsave Command Arguments

Command	Meaning
U	Inserts the contents of the save buffer into the text buffer.
0U	Clears the save buffer and reclaims the area for text.

The only argument the U command accepts is 0.

The contents of the save buffer are not destroyed by the Unsave command (only by the 0U command) and can be unsaved as many times as desired. If the Unsave command causes an overflow of the text buffer, the *?EDIT-F-Insufficient memory* error message prints, and the command does not execute.

For example:

```
*U$$
```

This command inserts the contents of the save buffer into the text buffer.

5.6.7.3 Macro – The Macro (M) command inserts a command string into the EDIT macro buffer. Table 5–14 lists the M commands and their meanings.

Table 5–14: M Command and Arguments

Command	Meaning
M/command string/	Stores the command string in the macro buffer.
0M or M//	Clears the macro buffer and reclaims the area for text.

The slash (/) represents the delimiter character. The delimiter is always the first character following the M command, and can be any character that does not appear in the macro command string itself.

Starting with the character following the delimiter, EDIT places the macro command string characters into its internal macro buffer until the delimiter is encountered again. At this point, EDIT returns to command mode. The macro command does not execute the macro string; it merely stores the command string so that the Execute Macro (EM) command can execute later. The Macro command does not affect the contents of the text or save buffers.

All characters except the delimiter are legal macro command string characters, including single ESCAPEs to terminate text commands. All commands, except the M and EM commands, are legal in a command string macro.

In addition to using the 0M command, you can type the M command immediately followed by two identical characters (assumed to be delimiters) and two ESCAPE characters to clear the macro buffer.

The following examples illustrate the use of the M command.

```
*M//$$
```

This command clears the macro buffer.

```
*M/GR0$-C1/$$
```

This command stores a macro to change R0 to R1.

NOTE

Be careful to choose infrequently used characters as macro delimiters; choosing frequently used characters can lead to errors. For example:

```
*M GMOV R0$=CADD R1$ $$
?EDIT-F-No file open for input
```

In this case, it was intended that the macro be GMOV R0$=CADD R1$, but since the delimiter character (the character following the M) is a space, the space following MOV is used as the second delimiter, terminating the macro. EDIT then returns an error when it interprets the R as a Read command.

5.6.7.4 Execute Macro – The Execute macro (nEM) command executes the command string previously stored in the macro buffer by the M command.

The syntax of the command is:

nEM

The argument *n* must be positive. The macro is executed *n* times and returns control to the next command in the original command string.

The following example uses the EM command.

```
*M/BGR0$-C1$/$$
*B1000EM$$
?EDIT-F-Search failed
*
```

This command sequence executes the macro stored in the previous example. EDIT prints an error message when it reaches the end of the buffer. (This macro changes all occurrences of R0 in the text buffer to R1.)

```
*IMOV PC,R1$2EMICLR @R2$$
*
```

This command inserts MOV PC,R1 into the text buffer and then executes the command in the macro buffer twice before inserting CLR @R2 into the text buffer.

5.6.7.5 Edit Version – The Edit Version (EV) command displays the version number of the editor in use on the console terminal.

The command is:

EV

This example displays the running version of EDIT:

```
*EV$$
V03.36
*
```

5.6.7.6 Upper- and Lower-Case Commands – If you have an upper- and lower-case terminal as part of your hardware configuration, you can take advantage of the two editing commands, Edit Lower (EL) and Edit Upper (EU).

When the editor is started with the EDIT command, upper-case mode is assumed – that is, all characters you type are automatically translated to upper case. To allow processing of both upper- and lower-case characters, enter the Edit Lower command. For example:

```
*EL$$
*i You can enter text and commands in UPPER and lower case.$$
*
```

The editor now accepts and echoes upper- and lower-case characters received from the keyboard, and prints text on the terminal in upper and lower case.

To return to upper-case mode, use the Edit Upper command:

```
*EU$$
```

Control also reverts to upper-case mode upon exit from the editor (with EX or CTRL/C).

Note that when you issue an EL command, you can enter EDIT commands in either upper or lower case. Thus, the following two commands are equivalent:

```
*GTEXT$=Cnew text$V$$

*gTEXT$=cnew text$v$$
```

The editor automatically translates (internally) all commands to upper case without reference to EL or EU.

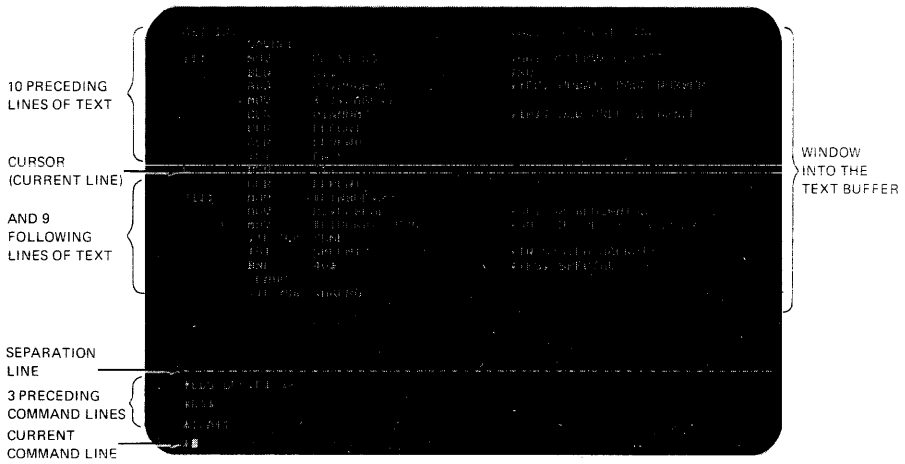
NOTE

When you use EDIT in EL mode, make sure that text arguments you specify in search commands have the proper case. The command GTeXt$, for example, will not match *TEXT, text,* or any combination other than *TeXt.*

5.7 Display Editor

In addition to all functions and commands mentioned thus far, the editor can use VT–11 and VS–60 display hardware that are part of the system configuration (GT40, GT44, DECLAB 11/40, DECLAB 11/34). The most obvious feature provided by this hardware is the use of the display screen rather than the console terminal for printing terminal input and output. Another feature is that the top of the display screen functions like a window into the text buffer. When all the features of the display editor are in use, a 12-inch screen displays text as shown in Figure 5–1.

Figure 5–1: Display Editor Format, 12-inch Screen



The major advantage is that you can now see immediately where the pointer is, because it appears between characters on the screen as a blinking L-shaped cursor. Remember that pressing the RET key causes both a carriage return and a line feed character to be inserted into the text. Note that if the pointer is placed between a carriage return and line feed, it appears in an inverted position at the beginning of the next line.

In addition to displaying the current line (the line containing the cursor), the 15 lines of text preceding the current line and the 14 lines following it are also in view on a 17-inch screen. Each time you execute a command string (with a double ESCAPE), EDIT refreshes this portion of the screen so that it reflects the results of the commands you just performed.

The lower section of the 17-inch screen contains eight lines of editing commands. The command line you are currently entering is last, preceded by the most recent command lines. A horizontal line of dashes separates this section from the text portion of the screen. As you enter new command lines, previous command lines scroll upward off the command section so that only eight command lines are ever in view.

A 12-inch screen displays 20 lines of text and 4 command lines.

5.7.1 Using the Display Editor

The display features of the editor are automatically invoked whenever the system scroller is in use (a monitor GT ON command is in effect) and you start the editor. However, if the system does not contain display hardware, the display features are not enabled.

If the system contains display hardware and you wish to use the screen during the editing session, you can activate it in one of two ways, whether or not the display is in use. (All editing commands and functions previously discussed in this chapter are valid for use.)

1. If the scroller is in use (the GT ON monitor command is in effect), EDIT automatically uses the screen for display of text and commands. However, it rearranges the scroller so that a window into the text buffer appears in the top two-thirds of the screen, while the bottom third displays command lines. This arrangement is shown in Figure 5–1.

   You can use the Edit Console command to return the scroller to its normal mode so that text and commands use the full screen, and the window is eliminated.

   The command is:

   EC

   This example uses the EC command:

   ```
   *BAEC2L$$
   ```

   This command lists the second and third lines of the current buffer on the screen; there is no window into the text buffer at this point.

   EDIT ignores subsequent EC commands if the window into the text buffer is not being displayed.

   To recall the window, use the Edit Display command:

   ED

   The screen is again arranged as shown in Figure 5–1.

2. Assume the scroller is not in use (the GT ON command is not in effect). When you call EDIT with the EDIT command, an asterisk appears on the console terminal. Use the ED command at this time to provide the window into the text buffer; however, commands continue to be echoed to the console terminal.

   When you use ED in this case, it must be the first command because you issue; otherwise, it becomes an illegal command (the memory used by the display buffer and code, amounting to over 600 words, is reclaimed as working space). You cannot use the display again until you load a fresh copy of EDIT.

   While the display of the text window is active, EDIT ignores ED commands.

   Typing the EC command clears the screen and returns all output to the console terminal.

NOTE

After completing an editing session that uses the ED command, clear the screen by typing the EC command or by returning to the monitor and using the monitor RESET command. Failure to do this may cause unpredictable results.

5.7.2 Immediate Mode

An additional mode is available to provide easier and faster interaction during the editing session. This mode is called immediate mode, which combines the most frequently used functions of the text and command modes — namely, repositioning the pointer and deleting and inserting characters.

You can use immediate mode only when the VT–11 display hardware is active and the editor is running. To enter immediate mode type two ESCAPEs (only) in response to the command mode asterisk:

```
*$$
```

The editor responds by displaying an exclamation point (!) on the screen.

The exclamation character remains on the screen as long as immediate mode is in effect.

Once you enter immediate mode, you can use only the commands in Table 5–15. Any other commands or characters are treated as text to be inserted. None of these commands echoes, but the text appearing on the screen is constantly refreshed and updated during the editing process.

To return control from immediate mode to normal command mode, type a single ESCAPE. The editor responds with an asterisk and you may proceed using all normal editing commands. (Immediate mode commands you type at this time will be accepted as command mode input characters.) To return control to the monitor from immediate mode, type ESCAPE to return to command mode, then type CTRL/C followed by two ESCAPEs.

Table 5–15: Immediate Mode Commands

Command	Meaning
CTRL/N	Advances the pointer (cursor) to the beginning of the next line (equivalent to A).
CTRL/G	Moves the pointer (cursor) to the beginning of the previous line (equivalent to –A).
CTRL/D	Moves the pointer (cursor) forward by one character (equivalent to J).
CTRL/V	Moves the pointer (cursor) back by one character (equivalent to –J).

(continued on next page)

Table 5–15: Immediate Mode Commands (Cont.)

Command	Meaning
RUBOUT or DELETE	Deletes the character immediately preceding the pointer (cursor) (equivalent to -D).
ESCAPE or ALTMODE	Single character returns control to command mode; double character directs control to immediate mode.
Any character other than those above	Inserts the character as text positioned immediately before the pointer (cursor) (equivalent to I).

5.8 EDIT Example

The following example illustrates the use of EDIT commands to change a program stored on the device DK:. Sections of the terminal output are coded by letter, and corresponding explanations follow the example.

```
A  .EDIT TEST1.MAC
   *R$$
   *
   *L$$
   ;TEST PROGRAM
   ;
   START:  MOV     #1000,SP        ;INITIALIZE STACK
B          MOV     #MSG,R0         ;POINT R0 TO MESSAGE
           JSR     PC,MSGTYP       ;PRINT IT
           HALT                    ;STOP
   MSG:    .ASCII/IT WORKS/
           .BYTE 15
           .BYTE 12
           .BYTE 0
C  *B$1J$5D$$
   *GPROGRAM$$
D  *0L$$
   ;PROGRAM*I TO TEST SUBROUTINE MSGTYP. TYPES
E  ;"THE TEST PROGRAM WORKS"
   ;ON THE TEMINAL$$
F  *F.ASCII/$$
   *$CTHE TEST PROGRAM WORKS$$
   *F.BYTE^X
G  *F.BYTE 0$V$$
           .BYTE 0
```

(continued on next page)

```
*I
          .END
$B/L$$
;PROGRAM TO TEST SUBROUTINE MSGTYP, TYPES
;"THE TEST PROGRAM WORKS"
;ON THE TERMINAL

START:  MOV     #1000,SP        ;INITIALIZE STACK
        MOV     #MSG,R0         ;POINT R0 TO MESSAGE
        JSR     PC,MSGTYP       ;PRINT IT
        HALT                    ;STOP
MSG:    .ASCII/THE TEST PROGRAM WORKS/
        .BYTE 15
        .BYTE 12
        .BYTE 0
        .END


*EX$$

.
```

A Calls the EDIT program and prints *. The input file is TEST1.MAC; the output file is TEST2.MAC. Reads the first page of input into the buffer.

B Lists the buffer contents.

C Places the pointer at the beginning of the buffer. Advances the pointer one character (past the ;) and deletes the TEST.

D Positions the pointer after PROGRAM and verifies the position by listing up to the pointer.

E Inserts text. Uses RUBOUT to correct typing error.

F Searches for .ASCII/ and changes IT WORKS to THE TEST PROGRAM WORKS.

G Types CTRL/X to cancel the P command. Searches for .BYTE 0 and verifies the location of the pointer with the V command.

H Inserts text. Returns the pointer to the beginning of the buffer and lists the entire contents of the buffer.

I Closes the input and output files after copying the current text buffer as well as the rest of the input file into the output file. EDIT returns control to the monitor.

5.9 EDIT Error Conditions

The editor prints an error message whenever it detects an error. EDIT checks for three general types of error conditions: (1) syntax errors, (2) exe-

cution errors, and (3) macro execution errors. This section describes the error message form for each type of error condition.

Before it executes any commands, EDIT first scans the entire command string for errors in command syntax, such as illegal arguments or an illegal combination of commands. If the editor finds an error of this type, it prints a message of this form:

```
?EDIT-F-Message; no command(s) executed
```

You should retype the command.

If a command string is syntactically correct, EDIT begins execution. Execution errors, such as buffer overflow or input and output errors, can still occur. In this case, EDIT prints a message of the form:

```
?EDIT-F-Message
```

EDIT executes all commands preceding the one in error. It does not execute the command in error or any commands that follow it.

When an error occurs during execution of a macro, EDIT prints a message of the form:

```
?EDIT-F-Message in macro;  no command(s) executed
```

or

```
?EDIT-F-Message in macro
```

Most errors are syntax errors. These are usually easy to correct before execution.

The *RT–11 System Message Manual* contains a complete list of the EDIT error messages, along with recommended corrective action for each error.

Part IV
Utility Programs

Part IV describes the utility programs available with RT–11. You can take advantage of nearly all of the capabilities of RT–11 by using the keyboard commands (described in Chapter 4), but it is the utility programs that actually perform many of the system's functions. For example, when you issue the CREATE command, the utility program DUP performs the create operation.

This part of the manual explains how to carry out utility operations, those not performed directly by the monitor, by running a specific utility program instead of using the keyboard monitor commands. It is not necessary to have an understanding of the material contained in Part IV in order to use the RT–11 system. However, the information in this part may be of interest to you if you have experience with a previous version of RT–11, or if you are a systems programmer and need to perform certain functions with the utility programs that are not available with the keyboard monitor commands.

Note that the syntax the Command String Interpreter requires for input and output specifications is different from the syntax you use to issue a keyboard monitor command. Chapter 6, the Command String Interpreter, describes the general syntax of the specification string that the system utility programs accept, and explains certain conventions and restrictions. Read this chapter carefully before you use any of the system utility programs directly, and bear in mind that there are many differences between issuing a monitor command and running a utility program. Chapters 7 through 18 describe the system utility programs themselves.

Chapter 6
Command String Interpreter

The Command String Interpreter (CSI) is part of the RT–11 that accepts a line of ASCII input, usually from the console terminal, and interprets it as a string of input specifications, output specifications, and options for use by a utility program. To call a utility program, respond to the dot (.) printed by the keyboard monitor by typing R followed by a program name and a carriage return. This example shows how to call the directory program (DIR):

```
.R DIR
*
```

The Command String Interpreter prints an asterisk (*) at the left margin of the terminal, indicating that it is ready to accept a list of specifications and options. The following section describes the syntax of the specifications and options you can enter.

6.1 CSI Syntax

Once you have started a system program, you must enter the appropriate information before any operation can be performed. You type a specification string in response to the prompting asterisk. The specifications are in the following general syntax:

output-filespecs/option = input-filespecs/option

A few system programs—EDIT and PATCH, for example—require you to enter this information differently. Complete instructions are provided in the appropriate chapters.

In all cases, the syntax for *output-filespecs* is:

dev:filnam.typ[n],...dev:filnam.typ[n]

The syntax for *input-filespecs* is:

dev:filnam.typ,...dev:filnam.typ

The syntax for */option* is:

/o[:oval] or /o[:dval].

where:

dev: represents either a logical device name or a physical device name, which is a two- or three-character name from Table 3–1.

If you do not supply a device name, the system uses device DK:. DK:, or whatever device you specify for the first file in a list of input or output files, applies to all the files in that input or output list, until you supply a different device name. For example:

*DT1:FIRST.OBJ,LP:=TASK.1,RK1:TASK.2,TASK.3

This command is interpreted as follows:

*DT1:FIRST.OBJ,LP:=DK:TASK.1,
RK1:TASK.2,RK1:TASK.3

File FIRST.OBJ is stored on device DT1:. File TASK.1 is stored on default device DK:. Files TASK.2 and TASK.3 are stored on device RK1:. Notice that file TASK.1 is on device DK:. It is the first file in the input file list and the system uses the default device DK:. Device DT1: applies only to the file on the output side of the command

filnam.typ — is the name of a file (consisting of one to six alphanumeric characters followed optionally by a period and a zero- to three-character file type). No spaces or tabs are allowed in the file name or file type. As many as three output and six input files are allowed. If you omit the dot and the file type, the system may apply a default file type that the program specifies.

[n] — is an optional declaration of the number of blocks (n) you need for an output file. *n* is a decimal number (up to 65,535) enclosed in square brackets immediately following the output *filnam.typ* to which it applies

/o[:oval] or /o[:dval]. — is one or more options whose functions vary according to the program you are using (refer to the option table in the appropriate chapter). *oval* is either an octal number or one to three alphanumeric characters (the first of which must be alphabetic) that the program converts to Radix–50 characters. *dval.* is a decimal number followed by a decimal point. You can use a minus sign (–) to denote negative octal or decimal numbers.

This manual uses the */o:oval* construction throughout, except for the keyboard monitor commands, where all values are interpreted as decimal (unless indicated otherwise) and the decimal point after a value is not necessary. However, the */o:dval.* format is always valid. Generally, these options and their associated values, if any, should follow the device and file name to which they apply.

If the same option is to be repeated several times with different values (for example, /L:MEB/L:TTM/L:CND) you

can abbreviate the line as /L:MEB:TTM:CND. You can mix octal, Radix–50, and decimal values

= is a delimiter that separates the output and input fields. You can use the < sign in place of the = sign. You can omit the separator entirely if there are no output files

NOTE

Except where noted, all numeric values you supply to the CSI must be in octal.

6.2 Prompting Characters

Table 6–1 summarizes the characters RT–11 prints either to indicate that the system is waiting for your response or to specify which job (foreground, system, or background) is producing output.

Table 6–1: Prompting Characters

Character	Explanation
. (dot)	The keyboard monitor is waiting for a command.
^	When the console terminal is being used as an input file, the uparrow (or circumflex) prompts you to enter information from the keyboard. Typing a CTRL Z marks the end-of-file.
>	If a foreground or system job is active, the > character identifies which job, foreground, system, or background, is producing the output that currently appears on the console terminal. Each time output from the background job is to appear, B> prints first, followed by the output. If the foreground job is to print output, F> prints first. If a system job is to print output, *jobname*> appears first, where *jobname* represents the name of the system job. See Section 3.6 for details on special function keys.
*	The current system utility program is waiting for a line of specifications and options.

Chapter 7
Peripheral Interchange Program (PIP)

The peripheral interchange program (PIP) is a file transfer and file maintenance utility program. You can use PIP to transfer files between any of the RT–11 devices (listed in Table 3–1) and to merge, rename, and delete files.

7.1 Calling PIP

To call PIP from the system device, respond to the keyboard monitor prompt (.) by typing:

```
R PIP<RET>
```

The Command String Interpreter prints an asterisk at the left margin of the terminal and waits for you to type a command string. If you enter only a carriage return at this point, PIP prints its current version number and prompts you again for a command string. You can type CTRL/C to halt PIP and return control to the monitor when PIP is waiting for input from the console terminal. You must type two CTRL/Cs to abort PIP at any other time. To restart PIP, type R PIP or REENTER followed by a carriage return in response to the monitor's dot.

Chapter 6, Command String Interpreter, describes the general syntax of the command line PIP accepts. You can type as many as six input file names, but only one output file name is allowed. If you specify a command involving random access devices for which the output specification is the same as the input specification, PIP does not move any files. However, it can change the creation dates on the files if you use /T, or it can rename the files if you use /R.

Because PIP performs file transfers for all RT–11 data formats (ASCII, object, and image), it does not assume file types for either input or output files. You must explicitly specify all file types where file types are applicable.

On random-access devices, such as disks and DECtape, PIP operations retain a file's creation date. If the file's creation date is 0, PIP gives it the current system date. However, in transfers to and from magtape and cassette, PIP always gives files the current system date.

You can use all variations of the wildcard construction for the input file specifications in the PIP command line (Section 4.2 describes wildcard usage). Output file specifications cannot contain embedded wildcards. If you use any wild character in an input file specification, the corresponding output file name or file type must be an asterisk. (The concatenate copy oper-

ation is an exception to this rule because it does not allow wildcards in the output specification.) The following example shows wildcard usage:

```
**.B=A%B.MAC
```

In the last example, the embedded percent character (%) represents any single, valid file name character. In the output file specification, the asterisk represents any valid file name.

The following command deletes all files with the file type .BAK (regardless of their file names) from device DK:.

```
**.BAK/D
```

The next command renames all files with a .BAK file type (regardless of file names) so that these files now have a .TST file type (maintaining the same file names).

```
**.TST=*.BAK/R
```

In most cases, PIP performs operations on files in the order in which they appear in the device directory. PIP ignores system files with the file type .SYS unless you also use the /Y option. PIP prints the error message *?PIP-W-No .SYS action* if you omit the /Y option on a command that would operate on .SYS files.

NOTE

> You cannot perform any operations that result in deleting a protected file. For example, you cannot transfer a file to a volume if a protected file with the same name already exists on the output volume.

PIP ignores all files with the file type .BAD unless you explicitly specify both the file name and file type in the command string. PIP does not print a warning message when it does not include .BAD files in an operation. Because of the way PIP handles .BAD files, you cannot use a wildcard (*.BAD) to perform any operation on them.

This example transfers all files, including system files, (regardless of file name or file type) from device DK: to device RK1:. It does not transfer .BAD files.

```
*RK1:*.*/Y=*.*
```

NOTE

> If you attempt to transfer files to a storage volume that has never been initialized with RT–11, a system failure may result.

7.2 Options

PIP options, summarized in Table 7–1, permit you to perform various operations with PIP. If you do not specify an option, PIP assumes that the operation is a file transfer in image mode. You can put command options at the end of the command string or type them after any file name in the string. Operations involving magtape are an exception, because the /M option is device-dependent and has a different meaning when you specify it on the input or output side of a command line. Type any number of options in a command line, as long as only one operation is represented. You can, however, combine copy and delete operations on one line.

Table 7–1: PIP Options

Option	Section	Explanation
/A	7.2.2.2	Copies files in ASCII mode, ignoring and discarding nulls and rubouts. It converts input file to 7-bit ASCII and treats CTRL/Z (32 octal) as the logical end-of-file on input (the default copy mode is image).
/B	7.2.2.3	Copies files in formatted binary mode (the default copy mode is image).
/C	7.2.2.4	Can be used with other options to include only files with the current date in the specified operation.
/D	7.2.3	Deletes input files from a specific device. Note that PIP does not automatically query before it performs the operation. If you combine /D with a copy operation, PIP performs the delete operation after the copy completes. This option is invalid in an input specification with magtape.
/E	7.2.7	Transfers files in a single- or small-disk system. PIP initiates the transfer, but pauses and waits for you to mount the volumes involved in the transfer.
/F	7.2.4.1	Protects files from deletion. Use with /R. Invalid for magtapes and cassettes.
/G	7.2.2.5	Ignores any input errors that occur during a file transfer and continues copying.
/K:n	7.2.2.6	Makes *n* copies of the output files to any sequential device, such as LP:, TT:, or PC:.
/M:n	7.2.1	You can use /M:n when I/O transfers involve either cassette or magtape. (See Section 7.2.1, Operations Involving Magtape and Cassette.)
/N	7.2.2.7	Does not copy or rename a file if a file with the same name exists on the output device. This option protects you from accidentally deleting a file. It is invalid for magtape and cassette in the output specification.
/O	7.2.2.8	Deletes a file on the output device if you copy a file with the same name to that device. The delete operation occurs before the copy operation. This option is invalid for magtape and cassette in the output specification.

(continued on next page)

Table 7–1: PIP Options (Cont.)

Option	Section	Explanation
/P	7.2.2.9	Copies or deletes all files except those you specify.
/Q	7.2.6	Use only with another operation. The /Q option causes PIP to print the name of each file to be included in the operation you specify. You must respond with a Y to include a particular file.
/R	7.2.4	Renames the file you specify. This operation is invalid for magtape and cassette.
/S	7.2.2.10	Copies files one block at a time.
/T	7.2.2.11	Puts the current date on all files you copy or rename, unless the current date is 0. This option is invalid for magtape and cassette; operations involving those devices always use the current date.
/U	7.2.2.12	Copies and concatenates all files you specify.
/W	7.2.5	Prints on the terminal a log of copy, rename, and delete operations.
/Y	7.2.2.13	Includes .SYS files in the operation you specify. You cannot modify or delete these files unless you use the /Y option.
/Z	7.2.4.2	Enables files for deletion, if they have been previously protected with /F. Use with /R. Invalid for magtapes and cassettes.

7.2.1 Operations Involving Magtape and Cassette

PIP handles magtape and cassette devices, which are sequential-access devices, differently from random-access devices, such as disks, diskettes, DECtape, and DECtape II. On magtape and cassette devices, files are stored serially, one after another, and there is no directory at the beginning of each device that lists the files and gives their location. Thus, you can access only one file at a time on each sequential-access device unit. Avoid commands that specify the same device unit number for both the input and output files — they are invalid.

The /M:n option makes operations that involve magtape and cassette more efficient. This option lets you specify different tape handling procedures for PIP to follow. The following sections outline the operations that involve magtape and cassette and describe the different procedures for using these devices that you can specify with the /M:n option. Remember that when you use the /M:n option, *n* is interpreted as an octal number. You must use *n.* (*n* followed by a decimal point) to represent a decimal number.

7.2.1.1 Using Cassette – The cassette is an inexpensive auxiliary storage medium. Note that DECtape II is not a cassette. Cassettes are used typically to store data such as text files or source programs. Clear plastic leader indicates the beginning-of-tape (BOT) and physical end-of-tape (EOT). A special sentinel file marks the end of current data and indicates where new data can begin. The /M:n option lets you position the tape a particular way, or rewind it, before beginning an operation. You can also use the /M:n option

to specify a special procedure for tape handling during cassette operations with PIP. The following operations are valid for use with cassettes: /A, /B, /C, /D, /G, /M, /P, /Q, /R, /S, /U, /W, and /Y. The following options are invalid with cassettes: /K, /N, /O, /R, /F, /Z, and /T. If you omit the /M:n option in a cassette operation, the cassette rewinds before each operation (using /M:0 has the same effect). The character, *n* in /M:n, represents a count of the number of files from the present position on the cassette. Note that the /M:n option has a different meaning for magtape (Section 7.2.1.2 describes how to use /M:n with magtape).

In copying to cassettes, /M:n functions as follows:

1. If *n* is 0:

   The cassette rewinds and PIP searches for the file you specify. If you specify more than one file, or if you use a wildcard in the file specification, the cassette rewinds before PIP searches for each file.

2. If *n* is a positive integer:

   PIP starts from the cassette's present position and searches for the file you specify. If PIP does not find the file by the time it reaches the *n*th file from its starting position, it uses the *n*th file for the read operation. Note that if PIP's starting position is not the beginning of the cassette, it is possible that PIP will not find the file you specify, even though it does exist on the tape.

3. If *n* is a negative integer:

   The cassette rewinds, then PIP follows the procedure outlined in step 2 above.

In writing to cassettes, /M:n functions as follows:

1. If *n* is 0:

   The cassette rewinds and PIP writes the file you specify starting at the logical end-of-tape (LEOT) position. PIP deletes any file it finds along the way that has the same name and file type as the file you specify.

2. If *n* is a positive integer:

   PIP starts from the cassette's present position and searches *n* files ahead, deleting along the way any file it finds that has the same name and file type as the file you specify. If it does not reach LEOT before it reaches the *n*th file from its starting position, it enters the file you specify over the *n*th file and deletes any files beyond it on the tape. If PIP reaches LEOT before it reaches the *n*th file, it writes the file you specify at the end-of-tape.

3. If *n* is a negative integer:

   The cassette rewinds, then PIP follows the procedure outlined in step 2 above.

If you are copying a file to cassette and reach the physical end-of-tape before the copy completes, PIP automatically continues the file on another cassette. The cassette device handler prints the *CTn: PUSH REWIND OR MOUNT NEW VOLUME* message. If you want to halt the copy operation at this point, push the cassette rewind button. The tape rewinds, PIP prints an error message, and then prompts you for a new command. However, if you want to continue the file on another cassette, remove the first cassette and put another initialized cassette in its place. The new cassette rewinds immediately. PIP then continues copying the file. The continued part of the file has the same file name and file type as the first part of the file, but PIP adds one to its sequence number to show that it is a continued file. Make sure you have a supply of initialized cassettes handy for cassette copy operations; you cannot interrupt the copy operation to initialize a cassette when PIP is waiting for a new volume. The following example shows a copy operation that fills one cassette and continues to another.

```
*CT1:*.*=RK:RT*.SYS,%%.SYS/Y/W/M:1
 Files copied:
RK:RT11SJ.SYS  to CT1:RT11SJ.SYS
CT1: PUSH REWIND OR MOUNT NEW VOLUME
RK:RT11FB.SYS  to CT1:RT11FB.SYS
RK:DT.SYS      to CT1:DT.SYS
RK:DP.SYS      to CT1:DP.SYS
RK:DX.SYS      to CT1:DX.SYS
RK:RF.SYS      to CT1:RF.SYS
RK:RK.SYS      to CT1:RK.SYS
RK:DM.SYS      to CT1:DM.SYS
RK:DS.SYS      to CT1:DS.SYS
RK:TT.SYS      to CT1:TT.SYS
RK:LP.SYS      to CT1:LP.SYS
RK:CR.SYS      to CT1:CR.SYS
RK:MT.SYS      to CT1:MT.SYS
RK:MM.SYS      to CT1:MM.SYS
RK:NL.SYS      to CT1:NL.SYS
RK:PC.SYS      to CT1:PC.SYS
RK:CT.SYS      to CT1:CT.SYS
RK:BA.SYS      to CT1:BA.SYS
*
```

A directory listing of the second cassette shows that the first file, RKMNFB.SYS, is continued from a previous tape. (The number of blocks in a cassette directory listing is not meaningful; it really represents the total of sequence numbers in the directory.)

```
*DIRECTORY CT1:
 15-Apr-79
RT11FB.SYS    1  15-Apr-79     DT .SYS    0  15-Apr-79
DP    .SYS    0  15-Apr-79     DX .SYS    0  15-Apr-79
RF    .SYS    0  15-Apr-79     RK .SYS    0  15-Apr-79
DM    .SYS    0  15-Apr-79     DS .SYS    0  15-Apr-79
TT    .SYS    0  15-Apr-79     LP .SYS    0  15-Apr-79
CR    .SYS    0  15-Apr-79     MT .SYS    0  15-Apr-79
MM    .SYS    0  15-Apr-79     NL .SYS    0  15-Apr-79
PC    .SYS    0  15-Apr-79     EL .SYS    0  15-Apr-79
CT    .SYS    0  15-Apr-79     BA .SYS    0  15-Apr-79
 18 Files, 1 Blocks
```

If you are reading a file from cassette that is continued on another volume, the cassette handler also prints the *CTn: PUSH REWIND OR MOUNT NEW VOLUME* message when it reaches the end of the first tape. To abort the operation, push the cassette rewind button; PIP then issues an error message and prompts for a new command. To continue the read operation, remove the first cassette and mount the second one in its place. The second cassette rewinds immediately and PIP searches for a file with the correct name and sequence number. PIP repeats the new volume message if it does not find the correct file. The following example copies a file that is continued on a second cassette.

```
*RK1:*.*=CT1:RT11FB.SYS/Y/W
 Files copied:
CT1: PUSH REWIND OR MOUNT NEW VOLUME
CT1:RT11FB.SYS to RK1:RT11FB.SYS
*
```

If you type a double CTRL/C during any output operation to cassette, PIP does not write a sentinel file at the end of the tape. Consequently, you cannot transfer any more data to the cassette unless you follow one of these recovery procedures:

1. Rewind the cassette. Then transfer all good files from the interrupted cassette to another cassette and initialize the interrupted cassette as the following example shows. Use any arbitrarily large number for /M:n.

   ```
   *CT1:*.*=CT0:DMPX.MAC,EXAMP.FOR/M:1000
   *^C

   .R DUP
   *CT0:/Z/Y
   *
   ```

2. Determine the sequential number of the file that was interrupted and use the /M:n option to enter a replacement file (either a new file or a dummy) over the interrupted file. PIP writes the replacement file and a sentinel file (LEOT) after it. The following example assumes the bad file is the fourth file on the cassette.

   ```
   *CT0:DUMMY.FIL=DT0:GLOBAL.MAC/M:-4
   *^C

   .DIRECTORY CT0:
    19-Apr-77
   DMPX   .MAC              0 19-Apr-77      MATCH .BAS      0 19-Apr-77
   EXAMP  .FOR              0 19-Apr-77      DUMMY .FIL      0 19-Apr-77
    4 Files, 0 Blocks
   ```

   A directory listing of the cassette shows three files and the replacement file.

To copy multiple files to a cassette with a wildcard command, use the following:

```
*CTn:*.*/M:1=dev:*.*
```

Continue to mount new cassettes in response to the *PUSH REWIND OR MOUNT NEW VOLUME* message. Do not abort the process by typing two consecutive CTRL/Cs, because continuation files may not be completed and no sentinel file will be written on the cassette.

To read multiple files from a cassette, use a command such as the following one. Use any arbitrarily large number for /M:n.

```
*dev:*.*=CTn:*.*/M:1000
```

Whenever PIP detects a continued volume, the *PUSH REWIND OR MOUNT NEW VOLUME* message appears, until the entire file has been copied (assuming that you mount each sequential cassette in response to each occurrence of the message). When PIP copies the final section of a continued file, it returns to command level. To copy the remaining files on that cassette, reissue the command:

```
*dev:*.*=CTn:*.*/M:1000
```

Repeat the process as often as necessary to copy all files. Do not abort the process by typing consecutive CTRL/Cs, because continuation files may not be completed.

7.2.1.2 Using Magtape – Magnetic tape is a convenient auxiliary storage medium for large amounts of data, and is often used as backup for disks. Reflective strips indicate the beginning and end of the tape. A special label (an EOF1 or EOV1 tape label) followed by two tape marks indicates the end of current data and also where new data can begin. The following PIP options are valid for use with magtape: /A, /B, /C, /G, /M, /P, /Q, /S, /U, /W, and /Y. These options are invalid with magtape: /K, /R, /F, /Z, and /T. The /M:n option lets you direct the tape operation; you can move the tape and perform an operation at the point you specify. Note that /D is invalid for input to magtape; /N and /O are invalid for output to magtape.

The /M:n option can be different for the output and input side of the command line. Since the option applies to the device and not to the files, you can specify one /M:n option for the output file and one for each input file. The /M:n option has a different meaning for cassette and magtape. Section 7.2.1.1 describes how to use /M:n with cassette.

Sometimes PIP begins an operation at the current position. To determine the current position, the magtape handler backspaces from its present position on the tape until it finds either an EOF indicator or the beginning of tape, whichever comes first. PIP then begins the operation with the file that immediately follows the EOF or BOT. The magtape handler also has a special procedure for locating a file with sequence number *n*:

1. If the file sequence number is greater than the current position, PIP searches the tape in the forward direction.

2. If the file sequence number is more than one file before the current position, or if the file sequence number is less than five files from the beginning-of-tape (BOT), the tape rewinds before PIP begins its search.

3. If the file sequence number is at the current position, or if it is one file past the current position, PIP searches the tape in the reverse direction.

Whenever you fetch or load a new copy of the magtape handler, the tape position information is lost. The "new" handler searches backward until it locates either BOT or a label from which it can learn the position of the tape. It then operates normally, according to steps 1, 2, and 3 described above.

If you omit the /M:n option, the tape rewinds between each operation. Using /M:0 has the same effect as omitting /M:n. When *n* is positive, it represents the file sequence number. When *n* is negative, it represents an instruction to the magtape handler.

In copying to magtapes, /M:n functions as follows:

1. If *n* is 0:

   The tape rewinds and PIP searches for the file you specify. If you specify more than one file, the tape rewinds before each search. If the file specification contains a wildcard, the tape rewinds only once and then PIP copies all the appropriate files.

2. If *n* is a positive integer:

   PIP goes to file sequence number *n*. If the file it finds there is the one you specify, PIP copies it. Otherwise, PIP prints the *?PIP-F-File not found* message. If you use a wildcard in the file specification, PIP goes to file sequence number *n* and then begins to search for matching files.

3. If *n* is –1:

   PIP starts the search at the current position. If the current position is not the beginning of the tape, PIP may not find the file you specify, even though it does exist on the tape.

In writing to magtapes, /M:n functions as follows:

1. If *n* is 0:

   The tape rewinds before PIP copies each file. PIP prints a warning message if it finds a file with the same name and file type as the input file and does not perform the copy operation.

2. If *n* is a positive integer:

   PIP goes to the file sequence number *n* and enters the file you specify. If PIP reaches logical end-of-tape (LEOT) before it finds file sequence number *n*, it prints the *?PIP-F-File sequence number not found* message. If you specify more than one file or if you use a wildcard in the file speci-

fication, the tape does not rewind before PIP writes each file, and PIP does not check for duplicate file names.

3. If *n* is –1:

   PIP goes to the LEOT and enters the file you specify. It does not rewind, and it does not check for duplicate file names.

4. If *n* is –2:

   The tape rewinds between each copy operation. PIP enters the file at LEOT or at the first occurrence of a duplicate file name.

If PIP reaches the physical end-of-tape before it completes a copy operation, it cannot continue the file on another tape volume. Instead, it deletes the partial file by backspacing and writing a logical end-of-tape over the file's header label. You must restart the operation and use another magtape.

If you type consecutive CTRL/Cs during any output operation to magtape, PIP does not write a logical end-of-tape at the end of the data. Consequently, you cannot transfer any more data to the tape unless you follow one of the following recovery procedures.

1. Transfer all good files from the interrupted tape to another tape and initialize the interrupted tape in the following manner:

```
*dev1:*.*=dev0:*.*
^C
.R DUP
*dev0:/Z/Y
```

2. Determine the sequential number of the file that was interrupted and use the /M:n construction to enter a replacement file (either a new file or a dummy) over the interrupted file. PIP writes the replacement file and a good LEOT after it. The following example assumes the bad file is the fourth file on the tape:

```
*dev0:file.new/M:4=file.dum
```

7.2.2 Copy Operations

PIP copies files in image, ASCII, and binary format. Other options let you change the date on the files, access .SYS files, combine files, and perform other similar operations. PIP automatically allocates the correct amount of space for new files in copy operations (except for concatenation). For block-replaceable devices, PIP stores the new file in the first empty space large enough to accommodate it. If an error occurs during a copy operation, PIP prints a warning message, stops the copy operation, and prompts you for another command. You cannot copy .BAD files unless you specifically type each file name and file type.

7.2.2.1 Image Mode – If you enter a command line without an option, PIP copies files onto the destination device in image mode. Note that you cannot

reliably transfer memory image files to or from paper tape, or to the line printer or console terminal. PIP can image-copy ASCII and binary data but it does not do any of the data checking described in Sections 7.2.2.2 and 7.2.2.3.

The following command makes a copy of the file named XYZ.SAV on device DK: and assigns it the name ABC.SAV. (Both files exist on device DK: after the operation.)

```
*ABC.SAV=XYZ.SAV
```

The next example copies from DK: all .MAC files whose names are three characters long and begin with A. PIP stores the resulting files on DX1:.

```
*DX1:*.*=A%%.MAC
```

7.2.2.2 ASCII Mode (/A) – Use the /A option to copy files in 7-bit ASCII mode. PIP ignores and eliminates nulls and rubouts during file transfer. PIP treats CTRL/Z (32 octal) as logical end-of-file if it encounters that character in the input file. The following command copies F2.FOR from device DK: onto device DT1: in ASCII mode and assigns it the name F1.FOR.

```
*DT1:F1.FOR=F2.FOR/A
```

7.2.2.3 Binary Mode (/B) – Use the /B option to transfer formatted binary files (such as .OBJ files produced by the assembler or the FORTRAN compiler and .LDA files produced by the linker). The following command, transfers a formatted binary file from the paper tape reader to device DK: and assigns it the name FILE.OBJ.

```
*DK:FILE.OBJ=PC:/B
```

When performing formatted binary transfers, PIP prints a warning if a checksum error occurs. If there is a checksum error and you did not use /G to ignore the error, PIP does not perform the copy operation. You cannot copy library files with the /B option. Copy library files in image mode.

7.2.2.4 Newfiles Option (/C) – The /C option copies only those files with the current date. Specify /C only once in the command line; it applies to all the file specifications in the entire command. The following command copies (in ASCII mode) all files named ITEM1 that also have the current date. It also copies the file ITEM2.MAC, if it has the current date, from DK: to DT2:. It combines all these files under the name NN3.MAC.

```
*DT2:NN3.MAC=ITEM1.*/C,ITEM2.MAC/A/U
```

The next command copies all files with the current date (except .SYS and .BAD files) from DK: to DX1:. This is an efficient way to back up all new files after a session at the computer.

```
*DX1:*.*=*.*/C
```

7.2.2.5 Ignore Errors Option (/G) – The /G option copies files, but ignores all input errors. This option forces a single-block transfer, which you can invoke at any other time with the /S option. Use the /G option if an input error occurred when you tried to perform a normal copy operation. The procedure can sometimes recover a file that is otherwise unreadable. If an error still occurs, PIP prints the *?PIP-W-Input Error* message and continues the copy operation.

The following command, copies the file TOP.SAV in image mode from device DT1: to device DK: and assigns it the name ABC.SAV.

```
*ABC.SAV=DT1:TOP.SAV/G
```

The next command copies files F1.MAC and F2.MAC in ASCII mode from device DT1: to device DT2:. This command creates one file with the name COMB.MAC, and ignores any errors that occur during the operation.

```
*DT2:COMB.MAC=DT1:F1.MAC,F2.MAC/A/G/U
```

7.2.2.6 Copies Option (/K:n) – The /K:n option directs PIP to generate *n* copies of the file you specify. The only legal output devices are the console terminal, the line printer, and paper tape punch. Normally, each copy of the file begins at the top of a page; copies are separated by form feeds.

```
*LP:=STOTLE.LST/K:3
```

This command, for example, prints three copies of the listing file, STOTLE.LST, on the line printer.

7.2.2.7 Noreplace Option (/N) – The /N option prevents execution of a copy or rename operation if a file with the same name as the output file already exists on the output device. This option is not valid when output is to magtape or cassette. The following example uses the /N option.

```
*DX0:CT.SYS=DK:CT.SYS/Y/N
?PIP-W-Output file found, no operation performed DK:CT.SYS
*
```

The file named CT.SYS already exists on DX0:, and the copy operation does not proceed.

7.2.2.8 Predelete Option (/O) – The /O option deletes a file on the output device if you copy a file with the same name to that device. PIP deletes the file on the output device before the copy operation occurs. Normally, PIP deletes a file of the same name after the copy completes. This option is not valid when output is to magtape or cassette. The following example uses the /O option.

```
*RK1:TEST1.MAC=DT2:TEST.MAC/O
```

If a file named TEST1.MAC already exists on RK1:, PIP deletes it before copying TEST.MAC from DT2: to TEST1.MAC on RK1:.

7.2.2.9 Exclude Option (/P) – The /P option directs PIP to transfer all files except the ones you specify.

```
*DT0:*.*=DX1:*.MAC/P
```

This command, for example, directs PIP to transfer all files from DX1: to DT0: except the .MAC files.

7.2.2.10 Single-Block Transfer Option (/S) – The /S option directs PIP to copy files one block at a time. On some devices, this operation increases the chances of an error-free transfer. You can combine the /S option with other PIP copy options. For example:

```
*RK1:TEST.MAC=RK0:TEST.MAC/S
```

PIP performs this transfer one block at a time.

7.2.2.11 Setdate Option (/T) – This option causes PIP to put the current date on all files it transfers, unless the current date is 0. Normally, PIP preserves the existing file creation date on copy and rename operations. This option is invalid for operations involving magtape and cassette because PIP always uses the current date for tape files. The following command puts the current date on all the files stored on device DK:.

```
**.*=*.*/Y/T
```

Note that the command shown above changes only the dates; PIP does not move or change the files in any other way.

7.2.2.12 Concatenate Option (/U) – To combine more than one file into a single file, use the /U option. This option is particularly useful when you want to combine several object modules into a single file for use by the linker or librarian. PIP does not accept wildcards on the output specification. Use the /B option with /U if you are concatenating object (.OBJ) files. The following examples show the /U option.

```
*DK:AA.OBJ=DT1:BB.OBJ,CC.OBJ,DD.OBJ/U/B
```

The command shown above transfers files BB.OBJ, CC.OBJ and DD.OBJ to device DK: as one file and assigns it the name AA.OBJ.

```
*DT3:MERGE.MAC=DT2:FILE2.MAC,FILE3.MAC/A/U
```

This command merges ASCII files FILE2.MAC and FILE3.MAC on DT2: into one ASCII file named MERGE.MAC on device DT3:.

7.2.2.13 System Files Option (/Y) – Use the /Y option if you need to perform an operation on system files (.SYS). For example:

```
**.*=DT3:*.*/Y
```

This command copies to device DK:, in image mode, all files (including .SYS files) from device DT3:.

7.2.3 Delete Operation (/D)

Use the /D option to delete one or more files from the device you specify. Note that PIP does not query you before it performs this operation, unless you use /Q. Remember to use the /Y option to delete .SYS files. You cannot delete .BAD files, unless you name each one specifically, including file name and file type. You can specify only six files in a delete operation unless you use wildcards. You must always indicate a file specification in the command line. A delete command consisting only of a device name (dev:/D) is invalid. The delete option is also illegal for magtape. The following examples illustrate the delete operation.

```
*FILE1.SAV/D
```

The command shown above deletes FILE1.SAV from device DK:.

```
*DX1:*.*/D
?PIP-W-No .SYS action
*
```

The command shown above deletes all files from device DX1: except those with a .SYS or .BAD file type. If there is a file with a .SYS file type, PIP prints a warning message to remind you that this file has not been deleted.

```
**.MAC/D
```

This command deletes all files with a .MAC file type from device DK:.

7.2.4 Rename Operation (/R)

Use the /R option to rename a file you specify as input, giving it the name you specify in the output specification. PIP prints an error message if the command specifications are not valid. Use the /Y option if you rename .SYS files. You cannot use /R with magtape or cassette.

The following examples illustrate the rename operation.

```
*DT1:F1.MAC=DT1:F0.MAC/R
```

The command shown above renames F0.MAC to F1.MAC on device DT1:.

```
*DX1:OUT.SYS=DX1:CT.SYS/Y/R
```

This command renames file CT.SYS to OUT.SYS.

The rename command is particularly useful when a file contains bad blocks. By giving the file a .BAD file type, you can ensure that the file permanently resides in that area of the device. Thus, the system makes no other attempts to use the bad area. Once you give a file a .BAD file type, you cannot move it during a compress operation. You cannot rename .BAD files unless you specifically indicate both the file name and file type.

7.2.4.1 File Protection Option (/F) – Use the /F option with /R to protect a file from deletion. The file in question appears as a protected file in the directory of the device in which that file resides. The letter *P* next to the block size number in the file's directory entry indicates the file is protected.

A sample command line for protecting the file SAVEME.TXT follows:

```
*SAVEME.TXT=SAVEME.TXT/R/F
```

If you copy a protected file, PIP does not copy the protected status.

7.2.4.2 File "Unprotection" Option (/Z) – Use the /Z option with /R to remove a file's protected status, enabling you to delete or change that file. When you use the /Z option, PIP removes the "P" from the file's directory entry.

In the following example, BOOT.MAC is enabled for deletion:

```
*BOOT.MAC=BOOT.MAC/R/Z
```

7.2.5 Logging Operation (/W)

When you use the /W option, PIP prints a list of all files copied, renamed, or deleted. The /W option is useful if you do not want to take the time to use the query mode (the /Q option, described in Section 7.2.6), but you do want a list of the files operated on by PIP.

PIP prints the log for an operation on the terminal under the command line. This example shows logging with the delete operation.

```
*DX1:*.*/D/W
?PIP-W-No .SYS action
 Files deleted:
DX1:TEST.MAC
DX1:FIX463.SAV
DX1:GRAPH.BAK
DX1:DMPX.MAC
DX1:MATCH.BAS
DX1:EXAMP.FOR
DX1:GRAPH.FOR
DX1:GLOBAL.MAC
DX1:PROSEC.MAC
DX1:EXAMP.MAC
*
```

7.2.6 Query Option (/Q)

Use the /Q option with another PIP operation to list all files and to confirm individually which of these files should be processed. Typing a Y (or any string that begins with Y) followed by a carriage return causes the named file to be processed; typing anything else excludes the file. The following example deletes files from DX1:.

```
*DX1:*.*/D/Q
 Files deleted:
DX1:FIX463.SAV?
DX1:GRAPH.BAK ? Y
DX1:DMPX.MAC  ?
DX1:MATCH.BAS ?
DX1:EXAMP.FOR ?
DX1:GRAPH.FOR ? Y
DX1:GLOBAL.MAC? Y
DX1:PROSEC.MAC? Y
DX1:KB.MAC    ?
DX1:EXAMP.MAC ?
*
```

7.2.7 Wait Option (/E)

If you have a single-disk system or a diskette system, you will find the /E option useful for copying operations. You use this option when you need to exchange storage volumes during a copy procedure. The general format of the command line follows.

*filespec/E = filespec

You can use any option with /E that is valid with your RT–11 configuration. You can not use wildcards as input. When you use /E, make sure that PIP is on your system volume.

When you use the /E option, PIP guides you through a series of steps in the process of completing the file transfer. After you enter the initial command string, PIP prints a message telling you what to do. After you complete each step, type a Y followed by a carriage return to proceed to the next step. When the transfer is complete, PIP prints a message instructing you to mount your system volume. After you have mounted the system volume, type a Y followed by a carriage return.

The sections that follow describe the procedure for single-volume and double-volume transfer.

7.2.7.1 Single-Volume Operation – If you want to transfer a file between two storage volumes, and you have only one drive for that type of storage volume, follow the procedure below.

1. Enter a command string according to this general syntax:

   *output-filespec/E = input-filespec

   where *output-filespec* represents the destination device and file specification, and *input-filespec* represents the source device and file specification.

2. PIP responds by printing the following message at the terminal.

   ```
   Mount input volume in <device>; Continue?
   ```

   <*device*> represents the device into which you are to mount your input volume. Type a Y followed by a carriage return after you have mounted your input volume.

3. PIP continues the copy procedure and prints the following message on the terminal:

```
Mount output volume in <device>;  Continue?
```

4. After you have removed your input volume from the device, mount your output volume and type Y followed by a carriage return.

5. Depending on the size of the file, PIP may repeat the transfer cycle (steps 2 and 3) several times before the transfer is complete. When the transfer is complete, PIP prints the following message if you had to remove the system volume from *<device>:*

```
Mount system volume in <device>;  Continue?
```

When you type a Y followed by a carriage return in response to the last instruction, you terminate the copy operation.

7.2.7.2 Double-Volume Operation – You can use the /E option for transferring files between two non-system volumes. The procedure for transferring files this way follows.

1. With your system volume mounted, enter a command string according to the following general syntax:

*output-filespec/E = input-filespec

where *output-filespec* represents the destination device and file specification, and *input-filespec* represents the source device and file specification.

2. After you have entered the command string, PIP responds with the message:

```
Mount input volume in <device>;  Continue?
```

Type a Y followed by a carriage return when you have mounted the input volume.

3. PIP then prints:

```
Mount output volume in <device>;  Continue?
```

Type a Y followed by a carriage return after you have mounted the output volume.

4. Unlike the single-volume transfer, the double-volume transfer involves only one cycle of mounting the input and output volumes. When the file transfer is complete, PIP prints the following message if you had to remove the system volume from *<device>:*

```
Mount system volume in <device>;  Continue?
```

When you type a Y followed by a carriage return in response to the last instruction, you terminate the copy operation.

Chapter 8
Device Utility Program (DUP)

The device utility program (DUP) is a device maintenance utility program that creates files on file-structured RT–11 devices (disks, single- and double-density diskettes, DECtape, DECtape II, magtape, and cassette). It can also extend files on certain file-structured devices (disks, single- and double-density diskettes, DECtape, and DECtape II) and it can compress, image copy, initialize, or boot RT–11 file-structured devices. DUP does not operate on non-file-structured devices (line printer, card reader, terminal, and paper tape).

8.1 Calling DUP

To call DUP from the system device, respond to the dot (.) printed by the keyboard monitor by typing:

```
R DUP <RET>
```

The Command String Interpreter prints an asterisk (*) at the left margin of the terminal and waits for you to type a command string. If you enter only a carriage return at this point, DUP prints its current version number and prompts you again for a command string. You can type CTRL/C to halt DUP and return control to the monitor when DUP is waiting for input from the console terminal. You must type two CTRL/Cs to abort DUP at any other time. Note that the /S, /T, and /C operations lock out the CTRL/C command until the operation completes; these three operations cannot be interrupted with CTRL/C. To restart DUP, type R DUP or REENTER in response to the monitor's dot. Chapter 6, Command String Interpreter, describes the general syntax of the command line that DUP accepts. DUP accepts only one input file specification and one output file specification in the command line.

8.2 Options

Certain options are available for use with DUP. These options are divided into two categories: (1) Action, and (2) Mode. Action options cause specific operations to occur. You can use these options alone or with valid mode options. Usually, you can specify only one action option at a time. Mode options modify action options. Table 8–1 illustrates which mode options you can use with a particular action option.

Table 8–1: DUP Options and Categories

Action	Mode
C	W,Y,G,E
D	W,Y
I	W,Y,G,E,F
K	W,F,H,G,E
O	W,Y
S	W,X,Y
T	W,Y
U	W,Y
V	W,Y
Z	W,B,N,R,V,Y,D

Note that /V can be either an action or a mode option, depending on how you use it.

You can use DUP action options to perform operation such as creating files, copying devices, scanning for bad blocks, performing a bootstrap operation, and initializing volumes. You can use the DUP mode options to modify the action options, where necessary. The following sections describe the various DUP options and give examples of typical uses. Table 8–2 summarizes the options you can use with DUP.

Table 8–2: DUP Options

Option	Section	Explanation
/B[:RET]	8.2.12.4	Use with /Z to write files with the file type .BAD over any bad blocks DUP finds on the disk to be initialized. Use :RET to retain through initialization all .BAD entries created by a previous /B.
/C	8.2.1	Use with /G to create a file on the device you specify; /G specifies the starting block number for the file to be created.
/D	8.2.12.5	Use with /Z to "uninitialize" a device. Use only if no files have been transferred to it since it was initialized.
/E:n	8.2.2	Specifies the ending block number for a read operation (used with the /I and /K options).
/F	8.2.3.1	Has two uses: use with the /K option to transfer the file name containing the bad block together with the relative block number of the bad block in the file. Or use with /I either to copy a file to an output device or copy a device to an output file.
/G:n	8.2.1	Specifies the starting block number for a read operation (on an input device) and the starting block number for a write operation (on an output device). *n* is an integer that represents a block number. Use this option with the /C, /I, and /K options.
/H	8.2.3	Use with the /K and /I options. Use with /K to read the bad block, write to the bad block, and then read it again. This operation does not destroy information already stored on the device. Use /H with /I to verify that the output is equal to the input.

(continued on next page)

Table 8–12: DUP Options (Cont.)

Option	Section	Explanation
/I	8.2.2	Copies the image of a disk to another disk or magtape or from magtape to disk. (Use with /G and /E if you want to specify block numbers.)
/K	8.2.3	Scans a device for bad blocks and outputs the octal address of the bad blocks to the output device. Use with /G and /E if you want to specify block numbers as boundaries for the scan.
/N:n	8.2.12.1	Use with /Z to set the number of directory segments you require if you do not want the default size; *n* is an integer in the range 1–37 (octal).
/O	8.2.4	Boots the device or file you specify.
/Q	8.2.5	Use with /O to boot a volume that is not RT–11, or is a pre-version 4 volume of RT–11.
/R[:RET]	8.2.12.3	Use with /Z to scan a device that supports bad block replacement for bad blocks. It then creates a replacement table on the disk for any bad blocks DUP finds. If you use :RET, DUP retains the replacement table that is already on the disk and does not pre-scan the disk for bad blocks.
/S	8.2.6	Compresses a disk (or DECtape) onto itself or onto another disk (or DECtape); the output device, if any, must be initialized.
/T:n	8.2.7	Extends an existing file by the number of blocks that *n* indicates.
/U[:xx]	8.2.8	Writes the bootstrap portion of the monitor file in blocks 0 and 2–5 of the target device. The optional argument, *xx*, represents the target system device name.
/V[:ONL]	8.2.9	Prints the user ID and owner name. Use it with /Z (as a mode option) to place a new user ID and owner name in block 1 of the initialized disk, or in the VOL1 header block on magtape (not applicable for cassette). Using /V:ONL with /Z changes only the ID and owner name, and does not initialize the device (not applicable for magtape or cassette).
/W	8.2.10	Use with any action option (but only one) to initiate an operation and then pause. This is useful on small, single-disk systems because it lets you replace the system device with another disk before performing an operation.
/X	8.2.6	Use with /S to inhibit automatic booting of the system device when it is compressed.
/Y	8.2.11	Use with /C, /I, /O, /S, /T, or /Z to inhibit the *dev:/xxxx Are you sure?* message and the *Foreground job loaded, CONTINUE?* message and ensure immediate execution of the operation.
/Z[:n]	8.2.12	Initializes the directory of the device you specify. The size of the directory defaults to the standard RT–11 size; use *n* to allocate extra directory words for each entry beyond the default.

8.2.1 Create Option (/C[/G:n])

The /C option creates a file with a specific name, location, and size on the block-replaceable device that you specify. This option is useful in recovering files that have been deleted. The /C option creates only a directory entry for

the file. It does not store any data in the file. You must specify both the file name and file type of the file to be created. The syntax of the command is:

filespec[n] = /C[/G:n]

where:

filespec[n]	represents the device file name, and file type of the file to be created; [*n*] is a decimal number representing the size in blocks of the file to be created. Note that the brackets here are part of the command; that is, they do not indicate *n* is optional. If you do not specify this number, DUP creates a one-block file
/G:n	represents the octal numeric value of the starting block of the file to be created. If you do not use /G:n, DUP creates the file in the first unused area large enough to contain the file

You can use the /C option to cover bad blocks on a disk by creating a file with a file type .BAD to cover the bad area.

You can also use /C to recover accidentally deleted files. In this case, use DIR to obtain a listing of the device. Use the /E and /Q options in DIR to list files, tentative files, empty areas, and the sizes of all areas. You can then assign a file name to the area that contains the data you lost.

You can also use DUP to set aside a file on a disk without performing any input or output operations on the file.

When you use the /C option, make sure that the area in which the file is to be created is empty (using the DIR /E and /Q options). If there are more blocks in the empty area than the file you are creating needs, DUP attempts to put the extra blocks in empty areas that are contiguous to the file you are creating. If there is not enough room in contiguous empty areas, the error message *?DUP-F-No room for file* prints, and DUP does not create the file. The /C option checks for duplicate file names. If the file name you specify already exists on the device, DUP issues an error message and does not create a second file with the same name.

The following example uses /C to create a file named FILE.MAC consisting of blocks 140, 141, and 142 on device DK1:.

```
*DK1:FILE.MAC[3]=/C/G:140
```

8.2.2 Image Copy Option (/I)

The /I option copies block for block from one volume to another. (This operation is not applicable for magtape or cassette.) The /I option is often used to copy one disk to another without changing the file structure or location of files on the device. For this purpose, it is an added convenience that you do not have to copy a boot block to the device. You can also copy disks that are not in RT–11 format, if they have no bad blocks. If DUP encounters a bad

block on either the input or output volume, it prints an error message. However, it retries the operation and performs the copy one block at a time. If only one error message prints, you can assume that the transfer completed correctly.

Qualifiers to the /I option let you:

1. Specify the blocks to be read from the input device and a starting block number for the write operation on the output device.
2. Copy a file to a device, or a device to a file, by specifying a file name with either the input or output device. For example, you can copy a diskette to an RL01 as a file, or a file on an RK05 to a diskette.

NOTE

When you use /F and you have specified a magtape or cassette as the input device, you must specify an input file name.

The syntax of the command is:

output-device: $\left\{\begin{matrix}\text{filename} \\ \text{A}\end{matrix}\right\}$ [/G:rn] = input-device[filename]/I[/G:rn/E:rn][/F]

where:

filename	represents the output file name of the input device, or (when specified with the input device) represents the input file name you are copying to the output device. If you specify an input file, use the dummy file name *A* with the output specification. Note that you can use a file name with either the input or output, but never with both
A	represents a dummy file name (required if the output device is not a magtape or cassette). Note that either *filename* or *A*, but not both, can be specified with the output device
/G:rn	when specified with the output device, represents the starting block number for the write operation. When specified with the input device, it represents the starting block number of the read operation
/E:rn	represents the ending block number on the input device for the read operation
/F	indicates that a file is involved in the transfer

The command string must include an input and an output specification; there is no default device. The /I operation does not copy to or from a device that has logical bad blocks. (Physical bad blocks can be logically replaced or

covered, as Sections 8.2.12.3 and 8.2.12.4 describe.) If one device is smaller than the other, DUP copies only the number of blocks of the smaller device.

You can copy blocks between disk and magtape. DUP stores the data on the tape, formatting it in 1K-word blocks. It is possible to store only one disk image on a magtape, regardless of the size of the tape.

NOTE

The /I option does not copy track 0 of diskettes. However, this restriction has no impact on any copy operations if your diskette was supplied by DIGITAL.

The following examples use the /I option. The file name A is not significant; it is a dummy file name required by the Command String Interpreter.

```
*RK1:A=RK0:/I
RK1:/Copy;  Are you sure?
```

The command shown above copies all blocks from RK0: to RK1:.

```
*RK1:A/G:501=RK0:/I/G:0/E:500
RK1:/Copy;  Are you sure?Y
```

The command shown above copies blocks 0–500 from RK0: to blocks 501 – 1000 on RK1:.

```
RK1:FLOPPY.BAK/F=DX0:/I
RK1:/Copy;  Are you sure?Y
```

The last command copies device DX0: to RK1: in a file named FLOPPY.BAK.

8.2.3 Bad Block Scan Option (/K)

Some mass storage volumes (disks, diskettes, DECtape, and DECtape II) have bad blocks, or they develop bad blocks as a result of age and use. You can use the /K option to scan a device and locate bad blocks on it. DUP prints the absolute block number of those blocks on the device that return hardware errors when DUP tries to read them. If you specify an output file, DUP prints the bad block report in that file. Remember that block numbers are octal and the first block on a device is block 0. If DUP finds no bad blocks, it prints an informational message. A complete scan of a volume takes from one to several minutes depending on the size of the volume. It does not destroy data that is stored on the device.

You can scan selected portions of a device by specifying beginning and ending block numbers. The syntax of this command is:

[filespec=]input-device:/K[/G:m[/E:n][/H]

where:

filespec	represents the output file specifications for the bad block report
/G:m	represents the block number of the first block to be scanned
/E:n	represents the block number of the last block to be scanned

If you specify only a starting block number, DUP scans from the block you specify to the end of the device.

If the device to be scanned has files on it, you can use /F with the /K option to print the name of the file containing the bad block and the relative block number within the file that is bad.

You can use /H with /K to read the bad block, write to the bad block, and then read it again. If the block is still bad, DUP reports a HARD error. If the block recovers, DUP reports a SOFT error. This procedure does not destroy data already stored on the device. Note that DIGITAL does not guarantee the integrity of data recovered from a soft bad block.

8.2.3.1 File Option (/F) – The file option serves two different purposes as a mode option, depending upon whether you use it with /K or with /I.

When you use /F with /K, DUP does a bad block scan and displays file names for each bad block it finds. DUP then prints a list of these bad block files along with their locations within the file. This list includes a relative block number of each bad block within the file and a report on whether each bad block is hard or soft. An example of such a list, along with the command line that generated it, follows.

```
*DX0:/K/F

    Block     Type                File          Block

000717     463.  Hard          NUMBER.PAS  000546     358.
000725     469.  Hard          ANTONY.MAC  000554     364.
000732     474.  Hard          CAESAR.MAC  000561     369.
000743     483.  Hard          < UNUSED >  000572     378.
000751     489.  Hard          < UNUSED >  000600     384.
000754     492.  Hard          < UNUSED >  000603     387.
?DUP-I-Bad blocks detected 6.
```

DUP outputs the following list if you use /F with /K on a disk that supports bad block replacement. In the column marked *Type,* DUP lists whether the bad block is replaced in the manufacturer's bad block replacement table or if it is hard or soft.

```
*DM1:/K/F
    Block          Type            File          Block
003055   1581.  Replaced      MSX    .SYS  000007      7.
003465   1845.  Replaced      DRV    .OBJ  000077     63.
037061  15921.  Replaced      < UNUSED >   010550   4456.
056106  23622.  Replaced      < UNUSED >   027575  12157.
056210  23688.  Replaced      < UNUSED >   027677  12223.
077521  32593.  Replaced      < UNUSED >   051210  21128.
143116  50766.  Replaced      < UNUSED >   043374  18172.
145337  51935.  Replaced      < UNUSED >   045615  19341.
?DUP-I-Bad blocks detected 8.
```

When you use /F with /I, you use it either to copy a file from an input device to an output device, or to copy an input device to an output file. Note that /I does not copy track 0 of diskettes. Note also that if you use a magtape or cassette for either the input or output device, you must specify a file name with the input device. For more information on /F refer to Section 8.2.2.

8.2.4 Boot Option (/O)

The /O option can perform two operations: (1) a hardware bootstrap of a specific device containing an RT–11 system and (2) a bootstrap of a particular RT–11 monitor file that does not affect the bootstrap blocks on the device. The command syntax for a device bootstrap is as follows:

dev:/O

This operation has the same results as a hardware bootstrap. Valid devices for the boot operation follow:

DT0:-DT7:	DD0:-DD1:
RK0:-RK7:	DL0:-DL3:
RF:	DY0:-DY1:
SY:	DM0:-DM7:
DK:	DS0:-DS7:
DP0:-DP7:	PD0:-PD1:
DX0:-DX1:	

Use the following syntax to boot a monitor without changing the bootstrap on the device:

dev:monitor-name/O

This makes it easy for you to switch from one monitor to another. Whether bootstrapping a specific monitor or a specific device, DUP checks to see if the bootstrap blocks are correctly formatted. If the boot operation you request is invalid, DUP prints an error message and waits for another command.

When you reboot with the /O option, you do not have to reenter the date and time of day with the monitor DATE and TIME commands. However, the clock does lose a few seconds during the reboot.

The following command reboots the RT–11 system under the single-job monitor:

```
*RK0:RT11SJ.SYS/O
RT-11SJ     V04.00
```

To boot a different monitor, for example the foreground/background monitor (for DX0:), type:

```
*DX0:RT11FB.SYS/O
```

8.2.5 Boot Foreign Volume Option (/Q)

Use the /Q option with /O to boot a volume that has a monitor other than the RT–11 version 4 monitor. Note that you must use /Q to boot any version 3B or earlier volume of RT–11. The following example boots an RT–11 version 3B volume.

```
*RK0:/O/Q
RT-11SJ V03B-00B
```

DUP does not retain the date and time when you use the /Q option.

8.2.6 Squeeze Option (/S)

Use the /S option to compress a volume (disk, diskette, DECtape) onto itself or onto another disk or DECtape. To do this, DUP moves all the files to the beginning of the device, producing a single, unused area after the group of files. The squeeze operation does not change the bootstrap blocks of a device. The output device you specify, if any, must be an initialized device. If you specify an output device, DUP does not query you for confirmation before it performs the operation. If you do not specify an output device, DUP prints the *Are you sure?* message and waits for your response before proceeding. You must type Y followed by a carriage return for the command to be executed. Since it is critical to perform an error-free squeeze operation, be sure to scan a device (with /K) before you use /S.

So you cannot reuse bad blocks, the /S option does not operate on files with .BAD file types. You can rename files containing bad blocks, giving them a .BAD file type, and therefore cause DUP to leave them in place when you execute a /S. DUP inserts files before and after .BAD files until the space between the last file it moved and the .BAD file is smaller than the next file to be moved. If an error occurs during a squeeze operation, DUP continues the operation, performing it one block at a time. If only one error message prints, you can assume that the operation completed correctly.

The syntax of the command is:

[output-device=]input-device/S

Do not use /S on the system device (SY:) when a foreground or system job is loaded. A *?DUP-F-Can't squeeze SY: while foreground loaded* error message results if you attempt this, and DUP ignores the /S operation. You must unload the foreground job before using the /S option.

NOTE

If you perform a compress operation on the system device, the system automatically reboots when the compress operation is completed. This operation takes place in order to prevent system crashes that can occur when a system file is moved.

You can use /X with /S to suppress the automatic reboot and leave DUP running. However, you should use /X only if you are certain that the monitor file will not move. Even then, you should reboot the system when the squeeze operation completes if the device handlers have moved.

The following examples use the /S option:

```
*SY:/S
SY:/Squeeze;  Are you sure?Y

RT-11SJ    V04.00
```

The command shown above compresses the files on the system device and reboots the system when the compress operation completes.

NOTE

If you compress your system volume, make sure DUP program within has the name DUP.SAV. If not, a system failure may occur.

```
*DT1:A=DT2:/S
```

This command transfers all the files from device DT2: to device DT1:, leaving DT2: unchanged. The file name A is not significant; it is a dummy file name required by the Command String Interpreter.

8.2.7 Extend Option (/T:n)

Use the /T option to extend the size of a file. The syntax of the command is:

filespec = /T:n

where:

filespec	represents the device, file name, and file type of the file to be extended
n	represents the number of blocks to add to the file

You can extend a file in this manner only if it is followed by an unused area at least *n* blocks long. Any blocks not required by the extend operation remain in the unused area.

The following example uses the /T option:

```
*DT1:ZYZ.TST=/T:100
```

This command assigns 100 more blocks to the file named ZYZ.TST on device DT1:.

8.2.8 Bootstrap Copy Option (/U[:xx])

In order to use a volume as a system volume, you must copy a bootstrap onto it. To do this, first make sure that the appropriate monitor file and handler are stored on the volume. For a diskette system, for example, check to see that the file DX.SYS is in the diskette directory. If it is, then you can copy the desired monitor onto the diskette, using the /U option. The option argument, *xx*, represents a target system device name. Use this argument when you are creating a bootable PDT volume if the current system is a PDP–11, and vice versa. Also you can use this argument when the current system is on an RX02 diskette and you wish to create a bootable RX01 diskette.

To copy a bootstrap for the single-job monitor on RK1:, for example, use the following procedure:

1. Obtain a formatted disk. (Most disks, diskettes, DECtape, and DECtape II volumes are formatted by the manufacturer. However, Chapter 18, FORMAT, does outline the procedure for reformatting RK05, RK06, RK07, RP02, RP03 disks, and RX01 and RX02 diskettes.)
2. Initialize the disk with /Z (see Section 8.2.12).
3. Copy files onto the disk.
4. Copy the monitor and RK05 handler, RK.SYS, onto the disk.
5. Copy the monitor bootstrap onto the disk with /U.

The following example shows how to initialize a diskette, copy files to it, and write a bootstrap onto the diskette:

```
*DX1:/Z/Y
```

The command shown above (step 2 of the procedure described above) initializes the diskette.

```
*DX1:A=DX0:/S
```

This command, which combines steps 3 and 4, squeezes all the files from DX0: onto DX1:.

```
*DX1:A=DX0:RT11FB.SYS/U
```

The last command (step 5) writes the bootstrap for the foreground/background monitor onto the bootstrap blocks (blocks 0 and 2–5) of DX1:. The file name A is not significant; it is a dummy file name required by the Command String Interpreter.

To create a bootable PDT–11/150 system diskette while running on a PDP–11, specify PD with the /U option. Likewise, if you are running on a PDT–11/150, and you wish to create a bootable PDP–11 system diskette,

specify DX (for single-density diskette) or DD (for DECtape II) with the /U option. The following command creates a bootable PDT–11/150 diskette while the current system is on a PDP–11:

```
*DX0:A=DX0:RT11SJ.SYS/U:PD
```

The next command creates a bootable PDP–11 diskette while the current system is on a PDT–11/150:

```
*PD0:A=PD0:RT11SJ.SYS/U:DX
```

8.2.9 Volume ID Option (/V[:ONL])

You can use the /V option as an action option to print the volume ID of a device or to change the volume ID. The syntax of the command is:

device:[/Z]/V[:ONL]

where:

device: is the device whose volume ID you want to display or change

If you specify only /V, DUP prints out on the console terminal the volume ID and owner name of the device you specify. If you specify /Z with /V, DUP initializes the device and prompts you for a new volume ID and owner name. If you specify /Z/V:ONL, DUP assumes you want only to change the volume ID and owner name and not initialize the device.

When you specify either /Z/V or /Z/V:ONL, DUP prompts you for a volume ID:

```
Volume ID?
```

Respond with a volume ID that is up to 12 characters long for a block-replaceable device. Terminate your response with a carriage return. DUP then prompts for an owner name:

```
Owner?
```

Respond with an owner name that is up to 12 characters long for a block-replaceable device. Terminate your response with a carriage return. DUP ignores characters you type beyond the legal length. You cannot change the volume ID of a magtape or cassette without initializing the entire tape. The /V:ONL command changes only the volume ID and owner name; it does not initialize the device. Section 8.2.12.2 describes how to use /V with the /Z option to initialize a device and write new volume identification on it.

The following example uses the /V:ONL option:

```
*RK1:/Z/V:ONL
RK0:/Volume ID change; Are you sure? Y
Volume ID? FORTRAN VOL
Owner?     Marcy
```

This command writes a new volume ID and owner name on device RK1:.

8.2.10 Wait for Volume Option (/W)

The /W option causes DUP to prompt you for the volumes to operate on, and waits for you to mount them. It is useful for single-disk systems or diskette systems. It is a mode option that you can use with any of the action options. However, you can perform only one operation at a time. The /W option initiates execution of a command, but then pauses and prints the message *Mount input volume in <device>; Continue?,* where *<device>* represents the device into which you mount the input volume. At this time you can remove the system disk (if necessary) and mount the disk on which you actually want the operation to take place. When the new disk is loaded, type a Y followed by a carriage return to execute the operation. DUP then prompts you for the input volume, if any. When the operation completes (except the /O operation, which boots the system), the *Mount system volume in <device>; Continue?* message prints. Replace the system device and type a Y followed by a carriage return. The asterisk (*) prompt prints, and DUP waits for you to enter another command. The following example uses the /W option:

```
*DX1:/K/F/W
Mount input volume in DX1; Continue? Y <RET>
?DUP-I-No bad blocks detected
Mount system volume in DX1; Continue? Y <RET>
*
```

This command directs DUP to scan the disk for bad blocks. During the first pause, the system disk is removed and another disk is mounted. A Y is typed and the scan operation executes. During the second pause, the system disk (on which DUP is stored) is replaced and another Y is typed. DUP prompts for another command. When you use /W, make sure that DUP is on the system volume.

8.2.11 Noquery Option (/Y)

Use the /Y option to suppress the query messages that some commands print. Certain options normally print the *Foreground job loaded, Continue?* message if a foreground job is loaded when you issue one of them (/C, /I, /O, /Q, /S, /T, and /Z). You must respond to the query message by typing Y followed by a carriage return for the operation to proceed. Some other options (/C, /I, /O, /S, /V, and /Z) print the *Are you sure?* message and wait for your response. If a foreground job is loaded and you specify one of these options, DUP combines the two query messages into one message and waits for your response. You can suppress all these messages and the pause associated with them by specifying /Y in the command string. Note, if you use /Y with /Z to initialize your system volume, the system ignores /Y.

8.2.12 Directory Initialization Option (/Z[:n])

You must initialize a device before you can store files on it. Use the /Z option to clear and initialize the directory of an RT–11 directory-structured device. The /Z option must always be the first operation you perform on a new device after you receive it, formatted, from a manufacturer. After you use /Z, there are no files in the directory.

The syntax of the command is as follows:

device:/Z[:n]

where:

device	represents the device you want to initialize
:n	is an octal integer (greater than or equal to 1) that represents the size increase, in words, of each directory entry. DUP adds this number to the default number of words allocated for each entry (valid only for directory-structured devices)

The size of the directory determines the number of files that can be stored on a device. The system allows a maximum of 72 files per directory segment, and 31 directory segments per device. Each segment uses two blocks of disk space. If you do not specify *n*, each entry is seven words long (for file name, creation date, and file position information). When you allocate extra words, the number of entries per directory segment decreases. The formula for determining the number of entries per directory segment is:

(512-7)/(number of extra words) + 7)

For example, if you use /Z:1, you can make 63 entries per segment. RT–11 does not normally support nonstandard directory formats, and DIGITAL does not recommend altering the directory format.

8.2.12.1 Changing Directory Segments (/N:n) – If you do not want the default directory size of the device, use /N with /Z to set the desired number of directory segments for entries in the directory. The syntax is as follows:

/N:n

In this option, *n* is an integer in the range 1–31 that represents the number of directory segments you want the directory to have.

Table 8–3 lists the default directory sizes, in segments, for RT–11-supported, directory-structured devices.

Table 8–3: Default Directory Sizes

Device	Size (decimal) of Directory in Segments
RK	16
DD	1
DT	1
RF	4
DS	4
DP	31
DX	1
DM	31
DY	4
DL	16
PD	1

If the default directory size for diskettes is too small for your needs, see the *RT–11 Installation and System Generation Guide* for details on increasing the default number of directory segments.

The following example initializes the directory on device RK1: and allocates six directory segments.

```
*RK1:/Z/N:6
RK1:/Initialize;  Are you sure?Y
```

8.2.12.2 Storing Volume ID (/V) – When you initialize a disk or magtape, DUP normally maintains the volume ID and owner name. If at initialization time you want to change the volume ID and owner name, use the /V option with /Z. For example, the following command initializes device RK1: and prompts you for a volume ID and owner name. Section 8.2.9 illustrates these prompts and shows how to use them.

```
*RK1:/Z/V
RK1:/Initialize;  Are you sure?Y
Volume ID? VOUCHERS
Owner? PAYABLES
```

8.2.12.3 Replacing Bad Blocks (/R[:RET]) – If you have RK06, RK07, RL01, or RL02 disks, use this option with /Z to scan a disk for bad blocks. If DUP finds any bad blocks, it creates a replacement table so that routine operations access good blocks instead of bad ones. Thus, the disk appears to have only good blocks. Note, though, that accessing this replacement table slows response time for routine input and output operations. With /R you have the option of deciding which bad blocks you want replaced if the number of bad blocks exceeds what can fit in the replacement table (replacement table overflow). The RK06 and RK07s support up to 32 (decimal) bad blocks in the replacement table; the RL01s and RL02s support up to 10.

When you use /R, DUP prints out a list of replaceable bad blocks as in the following sample:

```
    Block          Type
030722  12754.  Replaceable
115046  39462.  Replaceable
133617  46991.  Replaceable
136175  48253.  Replaceable
136277  48319.  Replaceable
136401  48385.  Replaceable
140405  49413.  Replaceable
146252  52394.  Replaceable
?DUP-I-Bad blocks detected 8.
```

If there is a replacement table overflow, DUP prompts you to indicate which blocks you want replaced as follows:

```
?DUP-W-Replacement table overflow
Type <RET>, 0, or nnnnnn (<RET>)
Replace block #
```

nnnnnn represents the octal block number of the block you want the system to replace.

After you enter a block number, DUP responds by repeating the *Replace block #* prompt. Type a 0 at any time if you do not want any more blocks replaced, and this will end prompting. DUP marks any blocks not placed in the replacement table as FILE.BAD.

If you enter a carriage return at any time, DUP places all bad blocks you have not entered into the replacement table, starting with the first on the disk, until the table is full. DUP assigns the name FILE.BAD to any remaining bad blocks and prompting ends.

If you use /Y with /R, the effect will be as if you entered a carriage return in response to the first Replace block prompt.

If you use :RET with /R, DUP initializes the volume and retains the bad block replacement table (and FILE.BAD files) created by the previous /R command.

Note that the monitor file cannot reside on a block that contains a bad sector error (BSE) if you are doing bad block replacement. If this condition occurs, a boot error results when you attempt to bootstrap the system. In case this happens, move the monitor.

8.2.12.4 Covering Bad Blocks (/B[:RET]) – To scan the volume for bad blocks and write files over them, use the /B option with /Z. For every bad block DUP encounters on the device, it creates a file called FILE.BAD to cover it. After the disk is initialized and the scan completed, the directory consists only of FILE.BAD entries that cover the bad blocks. If DUP finds a bad block in the boot block or the directory, it prints an error message and the disk is not usable.

If you specify :RET with /B, DUP will retain through initialization all FILE.BAD files created by a previous /B.

8.2.12.5 Restoring a Disk (/D) – Use /D to "uninitialize" a volume if you have not transferred any files to it since initialization. DUP will restore all files and directory entries that were present before the volume was initialized. This option is useful if you initialize a volume by mistake.

Note that /D does not restore boot blocks. Thus, if you use /D to restore a previously bootable volume, use the bootstrap copy option, /U[:xx], to make the volume bootable again.

Chapter 9
The Directory Program (DIR)

The directory program (DIR) performs a wide range of directory listing operations. It can list directory information about a specific device, either in summarized form — where only the number of files stored per segment is given — or in more detailed form — where file names, file types, creation dates, and other file information is given. DIR can organize its listings in several ways, such as alphabetically or chronologically.

9.1 Calling and Using DIR

To call DIR from the system device, respond to the dot (.) printed by the keyboard monitor by typing:

```
R DIR <RET>
```

The Command String Interpreter prints an asterisk at the left margin of the terminal and waits for you to enter a command string. If you enter only a carriage return in response to the asterisk, DIR prints its current version number. You can type CTRL/C to halt DIR and return control to the monitor when DIR is waiting for input from the console terminal. You must type two CTRL/Cs to abort DIR at any other time. To restart DIR, type R DIR or REENTER in response to the monitor's dot. Chapter 6, Command String Interpreter, describes the general syntax of the command line that DIR accepts. Unless otherwise indicated, numeric arguments are interpreted as octal. Remember to put a decimal point after a decimal number to distinguish it from an octal number. Some of the DIR options accept a date as an argument in the command line. The syntax for specifying the date is:

dd.:mmm:yy.

where:

dd. represents the day (a decimal integer in the range 1–31)

mmm represents the month (the first three characters of the name of the month)

yy. represents the year (a decimal integer in the range 73–99)

You can specify only one input device and one output device, but you can specify up to six file names on the input device. The default device for output is the terminal. The default file type for an output file is .DIR. The default device for input is DK:. If you omit the input specification completely, DIR uses DK:*.*. If you do not supply an option, DIR performs the /L operation. Note that wildcards are valid with DIR for the input specification only.

If you have selected timer support through the system generation process, but have not selected automatic end-of-month date advancement, make sure that you set the date at the beginning of each month with the DATE command. If you fail to set the date at the beginning of each month, DIR prints *-BAD-* in the creation date column of each file created beyond the end-of-month. (Note that you can eliminate a *-BAD-* entry by using the RENAME/SETDATE command after you have set the date.)

Directory listings normally print on the terminal in two columns. Read the entries across the columns, moving from left to right, one row at a time. Directory listings that are sorted, however, are an exception to this. (Sorted directories are produced by /A, /R, and /S options.) Read these listings by reading the left column from top to bottom, then reading the right column from top to bottom.

9.2 Options

You can perform many different directory operations by specifying options in the DIR command line. Table 9–1 summarizes the operations these options permit you to perform with DIR. The sections following the table describe the various DIR options and give examples; the options are arranged alphabetically in these sections.

Table 9–1: DIR Options

Option	Section	Explanation
/A	9.2.1	Lists the directory of the device you specify in alphabetical order by file name and type (this is the same as /S:NAM).
/B	9.2.2	Lists the directory of the device you specify, including file names and types, creation dates, starting block number and the number of blocks in each file. For magtape, the starting block number is the file sequence number. Note that DIR lists block numbers in decimal, unless you use the /O option.
/C:n	9.2.3	Lists the directory in *n* columns; *n* is an integer in the range 1–9. The default value is two columns for normal listings and five columns for abbreviated listings.
/D[:date]	9.2.4	Lists a directory containing only those files having the date you specify. If you do not supply a date, DIR uses the system's current date.
/E	9.2.5	Adds unused spaces and their sizes to the listing of the device directory. An empty space on a cassette directory represents a deleted file.
/F	9.2.6	Prints a five-column, short directory (file names and types only) of the device you specify.
/G	9.2.7	Lists the file you specify and all files that follow it in the directory. This option does not list any files that precede the file you specify.

(continued on next page)

Table 9–1: DIR Options (Cont.)

Option	Section	Explanation
/J[:date]	9.2.8	Prints a directory of the files created on or after the date you specify. If you do not supply a date, DIR uses the system's current date.
/K[:date]	9.2.9	Prints a directory of files created before the date you specify. If you do not supply a date, DIR uses the system's current date.
/L	9.2.10	Lists the directory of the device you specify, including the number of files, their dates, and the number of blocks each file occupies. (This is the default operation.)
/M	9.2.11	Lists a directory of unused areas of the device you specify.
/N	9.2.12	Lists a summary of the device directory.
/O	9.2.13	Similar to /L but lists the sizes and block numbers of the files in octal.
/P	9.2.14	Prints a directory of the device you specify, excluding the files you list.
/Q	9.2.15	Lists a directory of the device you specify, listing the file names and types, sizes, creation dates and starting block numbers of files that have been deleted and whose file name information has not been destroyed.
/R	9.2.16	Lists the files in the reverse order of the sort specified with /A or /S.
/S[:xxx]	9.2.17	Lists the directory of the device you specify in the order you specify; *xxx* indicates the order in which DIR sorts the listing (*xxx* can be DAT, NAM, POS, SIZ, or TYP).
/V[:ONL]	9.2.18	Lists the volume ID and owner name as part of the directory listing header. If you specify /V:ONL, DIR lists only the volume ID and owner name.

9.2.1 Alphabetical Option (/A)

The /A option lists the directory of the device you specify in alphabetical order by file name and type. Note that /A sorts numbers after letters. It has the same effect as the /S:NAM option. The following example lists the directory of device DX0: in alphabetical order.

```
*DX0:/A
 14-Dec-79
BUILD .MAC    100  06-Sep-79       SWAP  .SYS     25  05-Dec-79
DX    .SYS      3  06-Sep-79       SYSMAC.MAC     41  19-Nov-79
MYPROG.MAC     36P 12-Oct-79       TM    .MAC     25  27-Nov-79
RFUNCT.SYS      4  19-Nov-79       TT    .SYS      2  19-Nov-79
RT11SJ.SYS     67  19-Nov-79       VTMAC .MAC      7  19-Nov-79
 10 Files, 306 Blocks
 180 Free blocks
```

9.2.2 Block Number Option (/B)

The /B option adds the starting block number in decimal of all the files listed in a directory of the volume you specify. The following example lists the directory of device DX0:, including the starting block numbers of files.

```
*DX0:/B
 14-Dec-79
FSM    .MAC     31P 19-Nov-79  2955  BATCH .MAC    102P 19-Nov-79  2986
ELCOPY.MAC      8P 19-Nov-79  3088  ELINIT.MAC     15P 19-Nov-79  3096
ELTASK.MAC     15P 19-Nov-79  3111  ERROUT.MAC     48P 19-Nov-79  3126
ERRTXT.MAC      9P 19-Nov-79  3174  SYCND .BL       3P 19-Nov-79  3183
SYSTBL.BL       4P 19-Nov-79  3186  SYCND .DIS      5P 19-Nov-79  3190
SYSTBL.DIS      4P 19-Nov-79  3195  SYCND .HD       5P 19-Nov-79  3199
ABSLOD.SAV     48  15-Mar-76  3204  CHESS .SAV     40  17-Aug-75  3252
PETAL .SAV     36  11-Sep-75  3292  LAMP  .SAV     29  16-Mar-79  3328
WUMPUS.SAV     30  16-Mar-79  3357
 17 Files, 348 Blocks
 138 Free blocks
```

9.2.3 Columns Option (/C[:n])

The /C[:n] option lists the directory in the number of columns you specify. The argument, *n*, represents an integer in the range 1–9. If you do not use the /C:n option, DIR lists the directory in two columns for normal listings and five columns for abbreviated listings. The following command, for example, lists the directory of device DX1: in one column.

```
*DX1:/C:1
 29-Nov-79
SWAP   .SYS     25P 19-Nov-79
RT11SJ.SYS      67P 19-Nov-79
RT11FB.SYS      80P 19-Nov-79
RT11BL.SYS      64P 19-Nov-79
TT     .SYS      2P 19-Nov-79
DT     .SYS      3P 19-Nov-79
DP     .SYS      3P 19-Nov-79
 7 Files, 244 Blocks
 242 Free blocks
```

9.2.4 Date Option (/D[:date])

The /D[:date] option includes in the directory listing only those files having the date you specify. The default date is the system's current date. For example, the following command lists all the files that were created on August 13, 1979.

```
*DX0:/D:13.:AUG:79.
 15-Sep-79
RT11SJ.SYS      67P 13-Aug-79      RT11FB.SYS      80P 13-Aug-79
RT11BL.SYS      63P 13-Aug-79      DX     .SYS      3P 13-Aug-79
SWAP   .SYS     25P 13-Aug-79      TT     .SYS      2P 13-Aug-79
DP     .SYS      3P 13-Aug-79      DY     .SYS      4P 13-Aug-79
LP     .SYS      2P 13-Aug-79      PIP    .SAV     16  13-Aug-79
DUP    .SAV     41  13-Aug-79      RESORC.SAV     15  13-Aug-79
DIR    .SAV     17  13-Aug-79      RK     .SYS      3  13-Aug-79
EDIT   .SAV     19  13-Aug-79      DD     .SYS      5  13-Aug-79
SRCCOM.SAV      13  13-Aug-79      BINCOM.SAV     11  13-Aug-79
SLP    .SAV      9  13-Aug-79      SIPP   .SAV     14  13-Aug-79
 20 Files, 412 Blocks
 73 Free blocks
```

9.2.5 Entire Option (/E)

The /E option lists the entire directory including the unused areas and their sizes in blocks (decimal). The following example lists the entire directory of device DX1:, including unused areas. Use it to find free space before you extend a file (with CREATE or DUP/C).

```
*DX1:/E
 14-Dec-79
SWAP   .SYS     25P 23-Oct-79       RT11SJ.SYS     67P 23-Oct-79
RT11FB.SYS      80P 19-Nov-79       RT11BL.SYS     64P 19-Nov-79
TT     .SYS      2P 19-Nov-79       DT     .SYS     3P 19-Nov-79
DP     .SYS      3P 23-Oct-79       DX     .SYS     3P 19-Nov-79
DY     .SYS      4P 19-Nov-79       RF     .SYS     3P 19-Nov-79
RK     .SYS      3P 19-Nov-79       DL     .SYS     4P 23-Oct-79
DM     .SYS      5P 23-Oct-79       DS     .SYS     3P 19-Nov-79
DD     .SYS      5P 23-Oct-79       LP     .SYS     2P 23-Oct-79
LS     .SYS      2P 19-Nov-79       CR     .SYS     3P 19-Nov-79
MS     .SYS      9P 27-Nov-79       MTHD   .SYS     3P 23-Oct-79
DISMT1.COM      9P 27-Nov-79       MMHD   .SYS     4P 19-Nov-79
NUMBER.PAS      1  11-Dec-79       TONY   .AGP    14 17-Aug-79
NUM3   .LST     1  13-Dec-79       < UNUSED >    565
 25 Files, 322 Blocks
 164 Free blocks
```

9.2.6 FAST Option (/F)

The /F option lists only file names and file types, omitting file lengths and associated dates. For example, the following command lists only file names and types from device DX0:.

```
*DX0:/F
 16-Aug-79
DX     .SYS     PIP    .SAV     DIR    .SAV     DUP   .SAV      SWAP .SYS
RT11SJ.SYS      RT11FB.SYS      RT11BL.SYS      TT    .SYS      DT   .SYS
 10 Files, 312 Blocks
 174 Free blocks
```

9.2.7 Begin Option (/G)

The /G option lists the directory of the device you specify, beginning with the file you specify and including all the files that follow it in the directory. Usually, the disk you are using as a system device contains a number of files the operating system needs. These files include .SYS monitor files, .SAV utility program files, and various .OBJ, .MAC, and .BAK files. They are generally grouped together and usually listed at the beginning of a normal device directory. Files that you create and use, such as source files and text files, are also generally grouped together and follow the operating system files in the directory. If you specify the name of the last system file with the /G in the command line, DIR prints a directory of only those files that you created and stored on the device. The following command, for example, lists the last system file (CT.SYS) and all the user files that follow it.

```
*DX0:CT.SYS/G
 10-Dec-79
CT     .SYS      5  10-Aug-79       DIR    .SAV     17  03-Aug-79
RK     .SYS      3  13-Aug-79       EDIT   .SAV     19  03-Aug-79
STARTS.COM       1  27-Aug-79       DD     .SYS      5  19-Aug-79
SRCCOM.SAV      13  13-Aug-79       BINCOM.SAV      11  05-Oct-79
SLP    .SAV      9  13-Aug-79       SIPP   .SAV     14  05-Oct-79
 10 Files, 107 Blocks
 73 Free blocks
```

9.2.8 Since Option (/J[:date])

The /J[:date] option lists a directory of all files stored on the device you specify that were created on or after the date you supply. The default date is the system's current date. The following command lists all files on device DT0: that were created on or after 28 July 79.

```
*DT0:/J:28.:JUL:79.
 14-Dec-79
RT11SJ.SYS     67P 28-Jul-79       RT11FB.SYS     80P 02-Sep-79
RT11BL.SYS     63P 19-Aug-79       DX     .SYS     3P 10-Sep-79
SWAP   .SYS    25P 02-Sep-79       TT     .SYS     2P 15-Sep-79
SIPP   .SAV    14  02-Sep-79
 7 Files, 154 Blocks
 332 Free blocks
```

9.2.9 Before Option (/K[:date])

The /K[:date] option prints a directory of files created before the date you specify. The default date is the system's current date. The following command lists all files stored on device DX1: that were created before March 15, 1979.

```
*DX1:/K:15.:MAR:79.
 13-Apr-79
FORTRA.SAV   191  14-Mar-79       BASIC .SAV     51  25-Feb-79
 2 Files, 242 Blocks
 38 Free blocks
```

9.2.10 Listing Option (/L)

The /L option lists the directory of the device you specify. The listing contains the current date, all files and their associated creation dates, the number of blocks used by each file, total free blocks on the device (if disk or DECtape), the number of files listed, and the total number of blocks used by the files. File lengths, number of blocks and number of files are indicated as decimal values. For example, the following command lists on the line printer the directory for device DT1:.

```
*LP:=DT1:/L
```

The line printer output looks like this:

```
 20-Nov-79
RT11SJ.SYS     67P 03-Jul-79       RT11FB.SYS     80P 13-Aug-79
RT11BL.SYS     63P 15-Mar-79       DX     .SYS     3P 13-Aug-79
SWAP   .SYS    25P 13-Aug-79       TT     .SYS     2P 13-Aug-79
DP     .SYS     3P 13-Aug-79       DY     .SYS     4P 13-Aug-79
LP     .SYS     2P 20-Nov-79       PIP    .SAV    16  25-Jul-79
DUP    .SAV    41  26-Mar-79       RESORC.SAV    15  13-Aug-79
EDIT   .SAV    19  13-Aug-79       STARTS.COM     1  27-Aug-79
SIPP   .SAV    14  13-Aug-79
 15 Files, 413 Blocks
 73 Free blocks
```

9.2.11 Unused Areas Option (/M)

The /M option lists only a directory of unused areas and their size on the device you specify. For example, the following command lists all the unused areas on device RK0:.

```
*RK0:/M
 14-Dec-79
< UNUSED >     11                  < UNUSED >      2
< UNUSED >     26                  < UNUSED >     32
< UNUSED >      1                  < UNUSED >    525
< UNUSED >      0                  < UNUSED >    565
 0 Files, 0 Blocks
 1162 Free blocks
```

9.2.12 Summary Option (/N)

The /N option lists a summary of the device directory. The following command lists the summary of the directory for device DK:.

```
*/N
 14-Nov-79

    44 Files in segment 1

    46 Files in segment 4

    37 Files in segment 2

    34 Files in segment 5

    38 Files in segment 3

    16 Available segments, 5 in use

 199 Files, 3647 Blocks
 1115 Free blocks
```

9.2.13 Octal Option (/O)

The /O option is similar to the /L option, but lists the sizes (and starting block numbers if you use /B) of the files in octal. If the device you specify is a magnetic tape or cassette, DIR prints the sequence number in octal. For

example, the following command lists the directory of device DX0:, with sizes in octal.

```
*DX0:/O
 14-Dec-79 Octal
MYPROG.MAC      44P 12-Nov-79       TM     .MAC      31  27-Nov-79
VTMAC .MAC       7  18-Oct-79       SYSMAC.MAC      51  19-Nov-79
SWAP  .SYS      31  05-Sep-79       ANTON .MAC       4  19-Nov-79
RT11SJ.SYS     103  19-Nov-79       TT     .SYS      2  19-Nov-79
DX     .SYS      3  29-Aug-79       BUILD .MAC     144  19-Nov-79
 10 Files,    462 Blocks
    264 Free blocks
```

9.2.14 Exclude Option (/P)

The /P option lists a directory of all files on a volume, excluding those that you specify. You may specify up to six files.

```
*DX1:*.SAV/P
 29-Oct-79
RT11SJ.MAC     67P 06-Sep-79       RT11FB.MAC     80P 06-Sep-79
RT11BL.MAC     63P 06-Sep-79       DX     .MAC     3P 06-Sep-79
SWAP  .MAC     25P 06-Sep-79       TT     .MAC     2P 06-Sep-79
DP     .MAC     3P 06-Sep-79       DY     .MAC     4P 06-Sep-79
LP     .MAC     2P 06-Sep-79       RK     .MAC     3  06-Sep-79
STARTS.COM      1  27-Aug-79       DD     .MAC     5  06-Sep-79
 12 Files, 258 Blocks
 73 Free blocks
```

This command lists all files on device DX1: except .SAV files.

9.2.15 Deleted Option (/Q)

The /Q option lists a directory of the device you specify, listing the file names, types, sizes, creation dates, and starting block numbers in decimal of files that have been deleted but whose file name information has not been destroyed. The file names that print represent either tentative files or files that have been deleted. This can be useful in recovering files that have been accidentally deleted. Once you identify the file name and location, you can use DUP to rename the area. See Section 8.2.1 for this procedure.

```
*DISK.DIR=/Q
```

This command creates a file called DISK.DIR on device DK: that contains directory information about unused areas from device DK:.

Use the monitor TYPE command to read the file:

```
.TYPE DISK.DIR/LOG
 Files copied:
DK:DISK.DIR     to TT:
 12-Oct-79
EXAMPL.FOR     23  09-Sep-79       MTHD  .SMP      5  03-Sep-79
SCOPE .PIC      3  22-Sep-79
 0 Files, 0 Blocks
 0 Free blocks
```

9.2.16 Reverse Option (/R)

The /R option lists a directory in the reverse order of the sort you specify with the /A or /S option.

```
*DX0:/S:SIZ/R
 14-Dec-79
BUILD .MAC    100  06-Sep-79       TM     .MAC     25  27-Nov-79
RT11SJ.SYS     67  19-Nov-79       VTMAC  .MAC      7  19-Nov-79
SYSMAC.MAC     41  19-Nov-79       RFUNCT.SYS       4  19-Nov-79
MYPROG.MAC     36P 12-Oct-79       DX     .SYS      3  06-Sep-79
SWAP  .SYS     25  05-Dec-79       TT     .SYS      2  19-Nov-79
 10 Files, 306 Blocks
 180 Free blocks
```

This command lists the directory of device DX0: in reverse chronological order.

9.2.17 Sort Option (/S[:xxx])

The /S[:xxx] option sorts the directory of the specified device according to a three-character code you specify as *:xxx*. Table 9–2 summarizes the codes and their functions.

Table 9–2: Sort Codes

Code	Explanation
DAT	Chronological by creation date. Files that have the same date are sorted alphabetically by file name and file type
NAM	Alphabetical by file name. Files that have the same file name are sorted alphabetically by file type (this has the same effect as the /A option)
POS	According to the position of the files on the device. This is the same as using /S with no code
SIZ	Based on file size in blocks. Files that are the same size are sorted alphabetically by file name and file type
TYP	Alphabetical by file type. Files that have the same file type are sorted alphabetically by file name

The following examples illustrate the /S option.

```
*DX0:/S:DAT
 14-Dec-79
BUILD .MAC    100  06-Sep-79       SYSMAC.MAC     41  19-Nov-79
DX     .SYS     3  06-Sep-79       TT     .SYS      2  19-Nov-79
MYPROG.MAC     36P 12-Oct-79       VTMAC  .MAC      7  19-Nov-79
RFUNCT.MAC      4  19-Nov-79       TM     .MAC     25  27-Nov-79
RT11SJ.SYS     67  19-Nov-79       SWAP   .SYS     25  05-Dec-79
 10 Files, 306 Blocks
 180 Free blocks
```

(continued on next page)

```
*DX0:/S:NAM
 14-Dec-79
BUILD .MAC    100  06-Sep-79       SWAP  .SYS     25  05-Dec-79
DX    .SYS      3  06-Sep-79       SYSMAC.MAC     41  19-Nov-79
MYPROG.MAC     36P 12-Oct-79       TM    .MAC     25  27-Nov-79
RFUNCT.SYS      4  19-Nov-79       TT    .SYS      2  19-Nov-79
RT11SJ.SYS     67  19-Nov-79       VTMAC .MAC      7  19-Nov-79
 10 Files, 306 Blocks
 180 Free blocks


*DX0:/S:POS
 14-Dec-79
RT11SJ.SYS     67  19-Nov-79       BUILD .MAC    100  06-Sep-79
DX    .SYS      3  06-Sep-79       SYSMAC.MAC     41  19-Nov-79
MYPROG.MAC     36P 12-Oct-79       TM    .MAC     25  27-Nov-79
SWAP  .SYS     25  05-Dec-79       VTMAC .MAC      7  19-Nov-79
RFUNCT.SYS      4  19-Nov-79       TT    .SYS      2  19-Nov-79
 10 Files, 306 Blocks
 180 Free blocks


*DX0:/S:TYP
 14-Dec-79
BUILD .MAC    100  06-Sep-79       DX    .SYS      3  06-Sep-79
MYPROG.MAC     36P 12-Oct-79       RFUNCT.SYS      4  19-Nov-79
SYSMAC.MAC     41  19-Nov-79       RT11SJ.SYS     67  19-Nov-79
TM    .MAC     25  27-Nov-79       SWAP  .SYS     25  05-Dec-79
VTMAC .MAC      7  19-Nov-79       TT    .SYS      2  19-Nov-79
 10 Files, 306 Blocks
 180 Free blocks


*DX0:/S:SIZ
 14-Dec-79
TT    .SYS      2  19-Nov-79       SWAP  .SYS     25  05-Dec-79
DX    .SYS      3  06-Sep-79       MYPROG.MAC     36P 12-Oct-79
RFUNCT.SYS      4  19-Nov-79       SYSMAC.MAC     41  19-Nov-79
VTMAC .MAC      7  19-Nov-79       RT11SJ.SYS     67  19-Nov-79
TM    .MAC     25  27-Nov-79       BUILD .MAC    100  06-Sep-79
 10 Files, 306 Blocks
 180 Free blocks
```

9.2.18 Volume ID Option (/V[:ONL])

The /V option prints the volume identification and owner name as part of the directory listing header. The optional argument, :ONL, prints only the volume ID and owner name. You can combine /V with any other option.

The following example uses the /V option.

```
*DX:/V
 14-Dec-79
 Volume ID: BACKUP2
 Owner    : Marcy
SWAP  .SYS    25P 19-Nov-79      RT11SJ.SYS    67P 19-Nov-79
RT11FB.SYS    80P 19-Nov-79      RT11BL.SYS    64P 19-Nov-79
TT    .SYS     2P 19-Nov-79      DT    .SYS     3P 19-Nov-79
DP    .SYS     3P 19-Nov-79      DX    .SYS     3P 19-Nov-79
DY    .SYS     4P 19-Nov-79      RF    .SYS     3P 19-Nov-79
RK    .SYS     3P 19-Nov-79      DL    .SYS     4P 19-Nov-79
 12 Files, 271 Blocks
 215 Free blocks
```

The next example uses the :ONL argument.

```
*DXO:/V:ONL
Volume ID: RT11 V4
Owner    : Judith
```

Chapter 10
MACRO–11 Program Assembly

This chapter describes how to assemble MACRO–11 programs under the RT–11 operating system.

Output from the MACRO–11 assembler includes any or all of the following:

1. A binary object file—the machine-readable logical equivalent of the MACRO–11 assembly language source code
2. A listing of the source input file
3. A cross-reference file listing
4. A table of contents listing
5. A symbol table listing

To use the MACRO–11 assembler, you should understand how to:

1. Initiate and terminate the MACRO–11 assembler (including how to format command strings to specify files MACRO–11 uses during assembly)
2. Assign temporary work files to non-default devices, if necessary
3. Use file specification options to override file control directives in the source program
4. Use the small version of MACRO–11 for PDP–11 systems with 8K memory, if necessary
5. Interpret error messages

The following sections describe these topics.

10.1 Calling the MACRO–11 Assembler

To call the MACRO–11 assembler from the system device, respond to the system prompt (a dot printed by the keyboard monitor) by typing:

```
R MACRO <RET>
```

When the assembler responds with an asterisk (*), it is ready to accept command string input. (You can also call the assembler using the keyboard monitor MACRO command; see Chapter 4 for a description of this command.)

The assembler now expects a command string consisting of the following items, in sequence:

1. Output file specifications
2. An equals sign
3. Input file specifications

Format this command string as follows (punctuation is required where shown):

dev:obj,dev:list,dev:cref/s:arg = dev:sourcei,...,dev:sourcen/s:arg

where:

dev	is any legal RT–11 device for output; any file-structured device for input
obj	is the file specification of the binary object file that the assembly process produces; the *dev* for this file should not be TT or LP
list	is the file specification of the assembly and symbol listing that the assembly process produces
cref	is the file specification of the CREF temporary cross-reference file that the assembly process produces. (Omission of *dev:cref* does not preclude a cross-reference listing, however.)
/s:arg	is a set of file specification options and arguments (Section 10.4 describes these options and associated arguments.)
sourcei	is a file specification for a MACRO–11 source file or MACRO library file (These files contain the MACRO language programs to be assembled. You can specify as many as six source files.)

The following command string calls for an assembly that uses one source file plus the system MACRO library to produce an object file BINF.OBJ and a listing. The listing goes directly to the line printer.

```
*DK:BINF.OBJ,LP:=DK:SRC.MAC
```

All output file specifications are optional. The system does not produce an output file unless the command string contains a specification for that file.

The system determines the file type of an output file specification by its position in the command string. Use commas in place of files you wish to omit. For example, to omit the object file, you must begin the command string with a comma. The following command produces a listing, including cross-reference tables, but not binary object files.

```
*,LP:/C= (source file specification)
```

You need not include a comma after the final output file specification in the command string.

Table 10–1 lists the default values for each file specification.

Table 10–1: Default File Specification Values

File	Default Device	Default File Name	Default File Type
Object	DK:	Must specify	.OBJ
Listing	Same as for object file	Must specify	.LST
Cref	DK:	Must specify	.TMP
First source	DK:	Must specify	.MAC
Additional source	Same as for preceding source file	Must specify	.MAC
System MACRO Library	System device SY:	SYSMAC	.SML
User MACRO Library	DK: if first file, otherwise same as for preceding source file	Must specify	.MAC

10.2 Terminating the MACRO–11 Assembler

If you have typed R MACRO and received the asterisk prompt but have not yet entered the command string, you can terminate MACRO–11 control by typing CTRL/C once. After you have completed the command string (thus beginning an assembly) you can halt the assembly process at any time by typing CTRL/C twice. This returns control to the system monitor, and a system monitor dot prompt appears on the terminal.

To restart the assembly process, type R MACRO in response to the system monitor prompt. You can also restart using the REENTER command in most cases; however, the RT–11 system does not accept the REENTER command if the assembler is producing a cross-reference listing when you halt the assembly.

10.3 Temporary Work File

Some assemblies need more symbol table space than available memory can contain. When this occurs the system automatically creates a temporary work file called WRK.TMP to provide extended symbol table space.

The default device for WRK.TMP is DK. To cause the system to assign a different device, enter the following command:

```
.ASSIGN dev:  WF
```

The *dev* parameter is the physical name of a file-structured device. The system assigns WRK.TMP to this device.

10.4 File Specification Options

At assembly time you may need to override certain MACRO directives appearing in the source programs. You may also need to direct MACRO–11 on the handling of certain files during assembly. You can satisfy these needs by including special options in the MACRO–11 command string in addition to the file specifications. Table 10–2 lists the options and describes the effect of each.

The general format of the MACRO–11 command string is repeated below for your convenience:

dev:obj,dev:list,dev:cref/s:arg = dev:sourcei,...,dev:sourcen/s:arg

Table 10–2: File Specification Options

Option	Usage
/L:arg	Listing control, overrides source program directive .LIST
/N:arg	Listing control, overrides source program directive .NLIST
/E:arg	Object file function enabling, overrides source program directive .ENABL
/D:arg	Object file function disabling, overrides source program directive .DSABL
/M	Indicates input file is MACRO library file
/C:arg	Control contents of cross-reference listing
/P:arg	Specifies whether input source file is to be assembled during pass 1 or pass 2

The /M and /P options affect only the particular source file specification to which they are directly appended in the command string.

Other options are unaffected by their placement in the command string. The /L option, for example, affects the listing file, regardless of where you place it in the command string.

The following subsections describe how to use the file specification options.

10.4.1 Listing Control Options

Two options, /L:arg and /N:arg, pertain to listing control. By specifying these options with a set of selected arguments (see Table 10–3) you can control the content and format of assembly listings. You can override at assembly time the arguments of .LIST and .NLIST directives in the source program.

Figure 10–1 shows an assembly listing of a small program. This listing shows the more important listing features. It labels each feature with the mnemonic ASCII argument that determines its appearance on the listing; the argument SEQ, for instance, controls the appearance of the source line sequence numbers.

Specifying the /N option with no argument causes the system to list only the symbol table, the table of contents, and error messages.

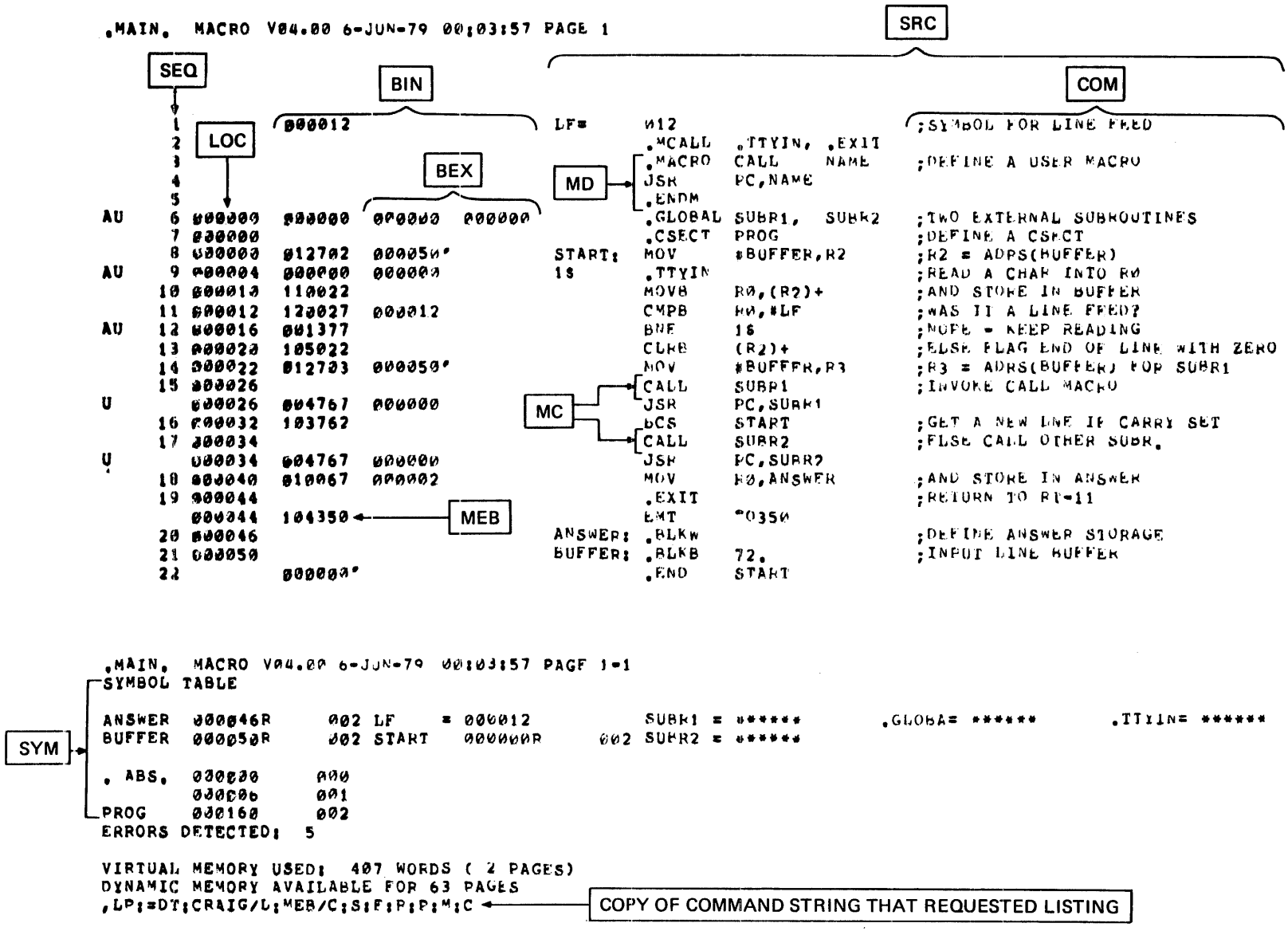
Specifying the /L option with no arguments causes the system to ignore .LIST and .NLIST directives that have no arguments.

The following example lists binary code throughout the assembly using the 132-column line printer format, and suppresses the symbol table listing.

```
*I,LP:/L:MEB/N:SYM=FILE
```

Table 10–3: Arguments for /L and /N Options

Argument	Default	Listing Control
SEQ	list	Source line sequence number
LOC	list	Address location counter
BIN	list	Generated binary code
BEX	list	Binary extensions
SRC	list	Source code
COM	list	Comment
MD	list	Macro definitions, repeat range expansion
MC	list	Macro calls, repeat range expansion
ME	no list	Macro expansions
MEB	no list	Macro expansion binary code
CND	list	Unsatisfied conditionals, .IF and .ENDC statements
LD	no list	List control directives with no arguments
TOC	list	Table of Contents
TTM	no list	132-column line printer format when not specified, terminal mode when specified
SYM	list	Symbol table




```
.MAIN.  MACRO V04.00 6-JUN-79 00:03:57 PAGE 1

        1  000012                   LF=     #12                     ;SYMBOL FOR LINE FEED
        2                           .MCALL  .TTYIN, .EXIT
        3                           .MACRO  CALL    NAME            ;DEFINE A USER MACRO
        4                           JSR     PC,NAME
        5                           .ENDM
AU      6  000000  000000  000000   .GLOBAL SUBR1,  SUBR2           ;TWO EXTERNAL SUBROUTINES
        7  000000                   .CSECT  PROG                    ;DEFINE A CSECT
        8  000000  012702  00005#'  START:  MOV     #BUFFER,R2      ;R2 = ADRS(BUFFER)
AU      9  000004  000000  000000   1$      .TTYIN                  ;READ A CHAR INTO R0
       10  000010  110022                   MOVB    R0,(R2)+        ;AND STORE IN BUFFER
       11  000012  120027  000012           CMPB    R0,#LF          ;WAS IT A LINE FEED?
AU     12  000016  001377                   BNE     1$              ;NOPE - KEEP READING
       13  000020  105022                   CLRB    (R2)+           ;ELSE FLAG END OF LINE WITH ZERO
       14  000022  012703  000050'         MOV     #BUFFER,R3      ;R3 = ADRS(BUFFER) FOR SUBR1
       15  000026                           CALL    SUBR1           ;INVOKE CALL MACRO
U          000026  004767  000000           JSR     PC,SUBR1
       16  000032  103762                   BCS     START           ;GET A NEW LINE IF CARRY SET
       17  000034                           CALL    SUBR2           ;ELSE CALL OTHER SUBR.
U          000034  004767  000000           JSR     PC,SUBR2
       18  000040  010067  000002           MOV     R0,ANSWER       ;AND STORE IN ANSWER
       19  000044                           .EXIT                   ;RETURN TO RT-11
           000044  104350                   EMT     #0350
       20  000046                   ANSWER: .BLKW                   ;DEFINE ANSWER STORAGE
       21  000050                   BUFFER: .BLKB   72.             ;INPUT LINE BUFFER
       22          000000'                  .END    START
```

```
.MAIN.  MACRO V04.00 6-JUN-79 00:03:57 PAGE 1-1
SYMBOL TABLE

ANSWER  000046R     002 LF     = 000012          SUBR1 = ******        .GLOBA= ******        .TTYIN= ******
BUFFER  000050R     002 START    000000R     002 SUBR2 = ******

. ABS.  000000      000
        000006      001
PROG    000160      002
ERRORS DETECTED:  5

VIRTUAL MEMORY USED:  407 WORDS ( 2 PAGES)
DYNAMIC MEMORY AVAILABLE FOR 63 PAGES
,LP:=DT:CRAIG/L:MEB/C:S:F:P:P:M:C
```

Figure 10–1: Sample Assembly Listing

10.4.2 Function Control Options

Two options, /E:arg and /D:arg, allow you to enable or disable functions at assembly time, and thus influence the form and content of the binary object file. These functions can override .ENABLE and .DSABL directives in the source program.

Table 10–4 summarizes the acceptable /E and /D function arguments, their normal default status, and the functions they control.

Table 10–4: Arguments for /E and /D Options

Argument	Default Mode	Function
ABS	Disable	Allows absolute binary output
AMA	Disable	Assembles all absolute addresses as relative addresses
CDR	Disable	Treats all source information beyond column 72 as commentary
CRF	Enable	Allows cross-reference listing. Disabling this function inhibits CREF output if option /C is active
FPT	Disable	Truncates floating point values (instead of rounding)
GBL	Disable	Treats undefined symbols as globals
LC	Disable	Allows lower-case ASCII source input
LSB	Disable	Allows local symbol block
PNC	Enable	Allows binary output
REG	Enable	Allows mnemonic definitions of registers

For example, if you type the following commands the system assembles a file while treating columns 73 through 80 of each source card as commentary.

```
.R PIP
*CARDS.MAC=CR:/A
*^C
.R MACRO
*,LP:=CARDS.MAC/E:CDR
```

Because MACRO–11 is a two-pass assembler, you cannot read the cards directly from the card reader or other non-file-structured device. You must use PIP (or the keyboard monitor COPY command) to transfer input to a file-structured device before beginning the assembly.

Use either the function control or listing control option and arguments at assembly time to override corresponding listing or function control directives in the source program. For example, assume that the source program contains the following sequence:

```
.NLIST MEB
  .
  . (MACRO references)
  .
.LIST MEB
```

In this example, you disable the listing of macro expansion binary code for some portion of the code and subsequently resume MEB listing. However, if you indicate /L:MEB in the assembly command string, the system ignores both the .NLIST MEB and the .LIST MEB directives. This enables MEB listing throughout the program.

10.4.3 Macro Library File Designation Option

The /M option is meaningful only if appended to a source file specification. It designates its associated source file as a macro library.

If the command string does not include the standard system macro library SYSMAC.SML, the system automatically includes it as the last source file in the command string.

When the assembler encounters an .MCALL directive in the source code, it searches macro libraries according to their order of appearance in the command string. When it locates a macro record whose name matches that given in the .MCALL, it assembles the macro as indicated by that definition. Thus if two or more macro libraries contain definitions of the same macro name, the macro library that appears leftmost in the command string takes precedence.

Consider the following command string:

*(output file specification)=ALIB.MAC/M,BLIB.MAC/M,XIZ

Assume that each of the two macro libraries, ALIB and BLIB, contain a macro called .BIG, but with different definitions. Then, if source file XIZ contains a macro call .MCALL .BIG, the system includes the definition of .BIG in the program as it appears in the macro library ALIB.

Moreover, if macro library ALIB contains a definition of a macro called .READ, that definition of .READ overrides the standard .READ macro definition in SYSMAC.SML.

10.4.4 Cross-Reference (CREF) Table Generation Option

A cross-reference (CREF) table lists all or a subset of the symbols in a source program, identifying the statements that define and use symbols.

10.4.4.1 Obtaining a Cross-Reference Table – To obtain a CREF table you must include the /C:arg option in the command string. Usually you include the /C:arg option with the assembly listing file specification. You can in fact place it anywhere in the command string.

If the command string does not include a CREF file specification, the system automatically generates a temporary file on device DK:. If you need to have a device other than DK: contain the temporary CREF file, you must include the *dev:cref* field in the command string.

If the listing device is magtape or cassette, load the handler for that device before issuing the command string, using the monitor LOAD command (described in Chapter 4).

A complete CREF listing contains the following six sections:

1. A cross reference of program symbols; that is, labels used in the program and symbols defined by a direct assignment statement.
2. A cross reference of register equate symbols. These normally include the symbols R0, R1, R2, R3, R4, R5, SP, and PC, unless the REG function has been disabled through a .DSABL REG directive or the /D:REG option. Also included are any other symbols that are defined in the program by the construct:

   symbol = % n

   where $0 \leq n \leq 7$ and *n* represents the register number.
3. A cross reference of MACRO symbols; that is, those symbols defined by .MACRO and .MCALL directives.
4. A cross reference of permanent symbols, that is, all operation mnemonics and assembler directives.
5. A cross reference of program sections. These symbols include the names you specify as operands of .CSECT or .PSECT directives. Also included are the default program sections produced by the assembler, the blank p-sect, and the absolute p-sect, . ABS.
6. A cross reference of errors. The system groups and lists all flagged errors from the assembly by error type.

You can include any or all of these six sections on the cross-reference listing by specifying the appropriate arguments with the /C option. These arguments are listed and described in Table 10–5.

Table 10–5: /C Option Arguments

Argument	CREF Section
S	User-defined symbols
R	Register symbols
M	MACRO symbolic names
P	Permanent symbols including instructions and directives
C	Control and program sections
E	Error code grouping

NOTE

Specifying /C with no arguments is equivalent to specifying /C:S:M:E. That special case excepted, you must explicitly request each CREF section by including its arguments. No cross-reference file occurs if the /C option is not specified, even if the command string includes a CREF file specification.

10.4.4.2 Handling Cross-Reference Table Files – When you request a cross-reference listing by means of the /C option, you cause the system to generate a temporary file, DK:CREF.TMP.

If device DK: is write-locked or if it contains insufficient free space for the temporary file, you can allocate another device for the file. To allocate another device, specify a third output file in the command string; that is, include a *dev:cref* specification. (You must still include the /C option to control the form and content of the listing. The *dev:cref* specification is ignored if the /C option is not also present in the command string.)

The system then uses the *dev:cref* file instead of DK:CREF.TMP and deletes it automatically after producing the CREF listing.

The following command string causes the system to use RK2:TEMP.TMP as the temporary CREF file.

```
*,LP:,RK2:TEMP.TMP=SOURCE/C
```

Another way to assign an alternate device for the CREF.TMP file is to enter the following command prior to entering R MACRO:

```
.ASSIGN dev:CF
```

This method is preferred if you intend to do several assemblies, because it relieves you from having to include the *dev:cref* specification in each command string. If you enter the *ASSIGN dev: CF* command, and later include a CREF specification in a command string, the specification in the command string prevails for that assembly only.

The system lists requested cross-reference tables following the MACRO assembly listing. Each table begins on a new page. (Figure 10–2 combines the tables to save space, however.)

The system prints symbols and also symbol values, control sections, and error codes, if applicable, beginning at the left margin of the page. References to each symbol are listed on the same line, left-to-right across the page. The system lists references in the form *p-l*; where *p* is the page in which the symbol, control section, or error code appears, and *l* is the line number on the page.

A number sign (#) next to a reference indicates a symbol definition. An asterisk (*) next to a reference indicates a destructive reference—that is, an operation that alters the contents of the addressed location.

Figure 10–2: Cross-Reference Table

```
.MAIN.  MACRO V04.00 6-JUN-79 00:03:57 PAGE S-1
CROSS REFERENCE TABLE (CREF V01-05 )


.GLOBA   1-6
.TTYIN   1-9
ANSWER   1-18*    1-20#
BUFFER   1-8      1-14      1-21#
LF       1-1#     1-11
START    1-8#     1-16      1-22
SUBR1    1-6      1-15
SUBR2    1-6      1-17



.MAIN.  MACRO V03.00 6-JUN-77 00:03:57 PAGE R-1
CROSS REFERENCE TABLE (CREF V01-05 )

PC       1-15*    1-17*
R0       1-10     1-11      1-18
R2       1-8*     1-12*     1-13*
R3       1-14*



.MAIN.  MACRO V03.00 6-JUN-77 00:03:57 PAGE M-1
CROSS REFERENCE TABLE (CREF V01-05 )

.EXIT    1-2#     1-19
.TTYIN   1-2#
CALL     1-3#     1-15      1-17



.MAIN.  MACRO V04.00 6-JUN-79 00:03:57 PAGE P-1
CROSS REFERENCE TABLE (CREF V01-05)

.BLKB    1-21
.BLKW    1-20
.CSECT   1-7
.END     1-22
.MACRO   1-3
.MCALL   1-2
BCS      1-16
BNE      1-12
CLRB     1-13
CMPB     1-11
EMT      1-19
JSR      1-15      1-17
MOV      1-8       1-14      1-18
MOVB     1-10



.MAIN.  MACRO V03.00 6-JUN-77 00:03:57 PAGE C-1
CROSS REFERENCE TABLE (CREF V01-05 )

         0-0
. ABS.   0-0
PROG     1-7



.MAIN.  MACRO V04.00 6-JUN-79 00:03:57 PAGE F-1
CROSS REFERENCE TABLE (CREF V01-05 )

A        1-6       1-9       1-12
U        1-6       1-9       1-12      1-15      1-17
```

10.4.5 Assembly Pass Option

The /P:arg option is meaningful only if appended to a source input file specification. You must specify either of two arguments with it: 1 or 2.

The specification /P:1 calls for assembly of the file during pass 1 only. Some files consist entirely of code that is completely assembled at the end of pass 1. By specifying /P:1 for these files, you can cause MACRO–11 to skip processing of these files through pass 2. In some cases this procedure can save considerable assembly time.

The specification /P:2 calls for assembly of the file during pass 2 only. (Note: Situations where the /P:2 option can be meaningfully employed are unusual.)

10.5 MACRO–11 8K Version

A subset version of MACRO–11, with file name MAC8K.SAV, is available for systems with 8K words of memory—that is, systems with insufficient memory to support operation of the full MACRO–11 assembler.

The full assembler (MACRO) requires approximately 10K words of memory, or must be operating on at least a 12K system using the single-job (SJ) monitor.

The subset version (MAC8K) requires approximately 6K words of memory, or must be operating on an 8K system using the baseline SJ monitor.

The subset version differs from the full assembler as follows:

1. All handlers must be resident (that is, loaded) before you call MAC8K.
2. The full assembler prints the input command string at the end of the listing; the subset version does not.
3. The subset version does not recognize the following items:
   a. The operation codes exclusive to PDP–11/45 and PDP–11/70
   b. The Commercial Instruction Set (CIS)
   c. The FLT2 and FLT4 floating point directives
4. The system device is the only file medium available under MAC8K.
5. The subset version does not support the cross-reference file and ignores attempts to obtain such a listing.
6. Assembly times of the subset version are noticeably longer.
7. The subset version operates only under control of the baseline single-job monitor (see the *RT–11 Installation and System Generation Guide*).
8. To get a program listing, specify a listing file name in the command line. You can then obtain the listing by using the PRINT or TYPE command.

10.6 MACRO–11 Error Codes

The MACRO–11 system prints diagnostic error codes as the first character of a source line on which the assembler detects an error. This error code identifies the type of error; for example, a code of M indicates a multiple definition of a label. Table 10–6 shows the error codes that might appear on an assembly listing. For detailed information on error code interpretation and debugging, see the *PDP–11 MACRO–11 Language Reference Manual.*

Table 10–6: MACRO–11 Error Codes

Error Code	Meaning
A	Addressing or relocation error. This message can be generated by any of the following: 1. a conditional branch instruction target that is too far above or below the current statement. Conditional branch targets must be within –128 to –127 (decimal) words of the instruction. 2. a statement that makes an illegal change to the current location counter. For example, a statement that forces the current location counter to cross a .PSECT boundary can generate this message. 3. a statement that contains an invalid address expression. For example, an absolute address expression that has a global symbol, relocatable value, or complex relocatable value can generate this message. The directives .BLKB, .BLKW, and .REPT must have an absolute value or an expression that reduces to an absolute value. 4. separate expressions in the statement that are not separated by commas. 5. a global definition error. If .ENABL GBL is set, MACRO–11 scans the symbol table at the end of the first pass and marks any undefined symbols as global references. If one of these symbols is subsequently defined in the second pass, a general addressing error occurs. 6. a global assignment statement that contains a forward reference to another symbol. 7. an expression that defines the value of the current location counter and contains a forward reference. 8. an illegal argument for an assembler directive 9. an unmatched delimiter or illegal argument construction
B	Instruction or word data are being assembled at an odd address. The system increments the location counter by 1, and continues.
D	A non-local label is defined more than once, specifically in an earlier statement.
E	The .END assembler directive at the end of the source input is missing. The system supplies a .END statement and completes the current assembly pass.
I	MACRO–11 has detected one or more illegal characters. A question mark (?) replaces each illegal character on the assembly listing, and MACRO–11 continues after ignoring the character.
L	An input line is longer than 132 characters. In particular, this error occurs when the expansion of a macro causes excessive substitution of real arguments for dummy arguments.

(continued on next page)

Table 10–6: MACRO–11 Error Codes (Cont.)

Error Code	Meaning
M	A label is the same as an earlier label (multiple definition of a label). For example, two labels whose first six characters are identical can generate this error.
N	A number is not in the current program radix. MACRO–11 processes this number as a decimal value.
O	Op-code error. Exceeding the permitted nesting level for conditional assemblies causes this error. Attempting to expand a macro that remains unidentified after a .MCALL search can also generate this message.
P	Phase error. The definition or value of a label differs from one assembler pass to the next, or a local symbol occurs more than once in a local symbol block.
Q	Questionable syntax. For example, missing arguments, too many arguments, or an incomplete instruction scan can generate this error message.
R	Register-type error. For example, if the source program attempts an invalid reference to a register, the assembler can generate this error message.
T	Truncation error. A number that generates more than 16 bits in a word, or an expression in a .BYTE directive or trap instruction, can cause this error message.
U	Undefined symbol. The assembler assigns the undefined symbol a constant zero value.
Z	Incompatible instruction. This message is a warning that the instruction is not defined for all PDP–11 hardware configurations.

Chapter 11
Linker (LINK)

The linker (LINK) converts object modules to a format suitable for loading and execution. If you have no previous experience with the linker, see the *Introduction to RT–11* for an introductory-level description of the linking process.

To make this chapter easy to use, the description that follows outlines the organization of this chapter.

Section 11.1, Overview of the Linking Process, explains:

- some of the terms used exclusively in this chapter
- the functions of the linker
- how the linker structures your program to prepare it for execution
- the communication links between modules within your program

Section 11.2, Calling and Using the Linker, describes how to invoke the linker from the system device and how to enter the command string. This section also provides a summary of the options you can use in the command string.

Section 11.3, Input and Output, lists and describes the files valid for input to and output from the linker. This section also explains how to use library files, and how the linker processes library files, which you create with the librarian utility (see Chapter 12).

Section 11.4, Creating an Overlay Structure, describes how to design and implement overlay structures for your programs. This section provides detailed descriptions and illustrations of how overlaid programs work and how they reside in memory. This section also explains how to create an overlay structure in extended memory.

Section 11.5, Options Description, lists and describes the options you can use with the linker.

Section 11.6, Linker Prompts, lists and explains the prompts the linker prints at the terminal after you enter a command line.

11.1 Overview of the Linker Process

A few of the terms used frequently within this chapter, along with their definitions, are listed below. Although the descriptions are brief, you can find more information on these terms in the *Introduction to RT–11* or the *RT–11 Software Support Manual*.

program section	A named, contiguous unit of code (instructions or data) that is considered an entity and that can be relocated separately without destroying the logic of the program. Also known as p-sect.
object module	The primary output of an assembler or compiler, which can be linked with other modules and loaded into memory as a runnable program. The object module is composed of the relocatable machine language code, relocation information, and the corresponding global symbol table defining the use of the symbols within the program. Also known as a module.
load module	A program in a format ready for loading and executing.
library file	A file containing one or more relocatable object modules that are routines that can be incorporated into other programs.
library module	A module from a library file.
root segment	The segment of an overlay structure that, when loaded, remains resident in memory during the execution of a program. Also known as the root.
overlay segment	A section of code treated as a unit that can overlay code already in memory and be overlaid by other overlay segments when called from the root segment or another overlay segment. Also known as an overlay.
global symbol	A global value or global label.
low memory	Physical memory from 0 to 28K words.
extended memory	Physical memory above the 28K word boundary.

11.1.1 What the Linker Does

When the linker processes the load modules, it performs the functions listed below.

- Relocates your program module and assigns absolute addresses
- Links the modules by correlating global symbols that are defined in one module and referenced in another
- Creates the initial control block for the linked program that the GET, R, RUN, SRUN, and FRUN commands use
- Creates an overlay structure, if specified, and includes the necessary run-time overlay handlers and tables
- Searches the library files you specify to locate unresolved global symbols

- Produces a load map, if specified, that shows the layout of the load module
- Produces a symbol table definition file, if specified

The linker requires two passes over the input modules. During the first pass it constructs the symbol table, which includes all program section names and global symbols in the input modules. Next, the linker scans the library files to resolve undefined global symbols. It links only those modules that are required to resolve undefined global symbols. During the second pass, the linker reads in object modules, performs most of the functions listed above, and produces the load module.

The linker runs in a minimal RT–11 system of 8K words of memory; any additional memory is used to facilitate linking and to extend the size of the symbol table. The linker accepts input from any random-access volume on the system; there must be at least one random-access volume (disk, diskette, DECtape, or DECtape II) for memory image or relocatable format output.

11.1.2 How the Linker Structures the Load Module

When the linker processes the assembled or compiled object modules, it creates a load module in which it has assigned all absolute addresses, has created an absolute section, and has allocated memory for the program sections.

11.1.2.1 Absolute Section — The absolute section is often called the ASECT because the assembler directive .ASECT allows information to be stored there. The absolute section appears in the load map with the name . ABS., and is always the first section in the listing. The absolute section typically ends at address 1000 (octal) and contains the following:

- a system communication area
- hardware vectors
- user stack

The system communication area resides in locations 0–377, and contains data the linker uses to pass program control parameters and a memory usage bitmap. Section 11.3.3 provides a detailed description of each location in the system communication area.

The stack is an area that a program can use for temporary storage and subroutine linkage. General register 6, the stack pointer (SP), references the stack.

11.1.2.2 Program Sections — The program sections (p-sects) follow the absolute section. The set of attributes associated with each p-sect controls the allocation and placement of the section within the load module. The p-sect, as the basic unit of memory for a program, has:

- a name by which it can be referenced
- a set of attributes that define its contents, mode of access, allocation, and placement in memory
- a length that determines how much storage is reserved for the p-sect

You create p-sects by using a COMMON statement in FORTRAN, or the .PSECT (or .CSECT) directive in MACRO. You can use the .PSECT (or .CSECT) directive to attach attributes to the section. Note that the attributes that follow the p-sect name in the load map are not part of the name; only the name itself distinguishes one p-sect from another. You should make sure, then, that p-sects of the same name that you want to link together also have the same attribute list. If the linker encounters p-sects with the same name that have different attributes, it prints a warning message and uses the attributes from the first time it encountered the p-sect.

Program Section Attributes

The linker collects from the input modules scattered references to a p-sect and combines them in a single area of the load module. The attributes, which are listed in Table 11–1, control the way the linker collects and places this unit of storage.

Table 11–1: P-sect Attributes

Attribute	Value	Explanation
access-code*	RW	Read/Write — data can be read from, and written into, the p-sect.
	RO	Read Only — data can be read from, but cannot be written into, the p-sect.
type-code	D	Data — the p-sect contains data, concatenated by byte.
	I	Instruction — the p-sect contains either instructions, or data and instructions, concatenated by word.
scope-code	GBL	Global — the p-sect name is recognized across segment boundaries. The linker allocates storage in the root for the p-sect from references outside the defining overlay segment. If the p-sect is referenced only in one segment, that p-sect has space allocated in that segment only.
	LCL	Local — the p-sect name is recognized only within each individual segment. The linker allocates storage for the p-sect from references within the segment only.
reloc-code	REL	Relocatable — the base address of the p-sect is relocated relative to the virtual base address of the program.
	ABS	Absolute — the base address of the p-sect is not relocated. It is always 0.
alloc-code	CON	Concatenate — all allocations to a given p-sect name are concatenated. The total allocation is the sum of the individual allocations.
	OVR	Overlay — all allocations to a given p-sect name overlay each other. The total allocation is the length of the longest individual allocation.

* Not supported

The scope-code is meaningful only when you define an overlay structure for the program. In an overlaid program, a global section is known throughout the entire program. Object modules contribute to only one global section of the same name. If two or more segments contribute to a global section, then the linker allocates that global section to the root segment of the program. In contrast to global sections, local sections are only known within a particular program segment. Because of this, several local sections of the same name can appear in different segments. Thus, several object modules contributing to a local section do so only within each segment. An example of a global section is named COMMON in FORTRAN. An example of a local section is the default blank section for each macro routine.

The alloc-code determines the starting address and length of memory allocated by modules that reference a common p-sect. If the alloc-code indicates that such a p-sect is to be overlaid, the linker stores the allocations from each module starting at the same location in memory. It determines the total size from the length of the longest reference to the p-sect. Each module's allocation of memory to a location overwrites that of a previous module. If the alloc-code indicates that a p-sect is to be concatenated, the linker places the allocations from the modules one after the other in the load module; it determines the total allocation from the sum of the lengths of the contributions.

Any data (D) p-sect that contains references to word labels must start on a word boundary. You can do this by using the .EVEN assembler directive at the end of each module's concatenated p-sect. (If you do not do this, the program may fail to link, printing the message *?LINK-F-Word relocation error.*)

The allocation of memory for a p-sect always begins on a word boundary. If the p-sect has the D (data) and CON (concatenate) attributes, all storage that subsequent modules contribute is appended to the last byte of the previous allocation. This occurs whether or not that byte is on a word boundary. For a p-sect with the I (instruction) and CON attributes, however, all storage that subsequent modules contribute begins at the nearest following word boundary.

The .CSECT directive of MACRO is converted internally by both MACRO and the linker to an equivalent .PSECT with fixed attributes. An unnamed CSECT (blank section) is the same as a blank PSECT with the attributes RW, I, LCL, REL, and CON.

A named CSECT is equivalent to a named PSECT with the attributes RW, I, GBL, REL, and OVR. Table 11–2 shows these sections and their attributes.

Table 11–2: Section Attributes

Section	access-code	type-code	scope-code	reloc-code	alloc-code
CSECT	RW	I	LCL	REL	CON
CSECT name	RW	I	GBL	REL	OVR
ASECT (. ABS.)	RW	I	GBL	ABS	OVR
COMMON/name/	RW	D	GBL	REL	OVR
VSECT (. VIR.)	RW	D	GBL	REL	CON

The names assigned to p-sects are not considered to be global symbols; you cannot reference them as such. For example:

```
MOV     #PNAME,R0
```

This statement, where PNAME is the name of a section, is invalid and generates the undefined global error message if no global symbol of PNAME exists. A name can be the same for both a p-sect name and a global symbol. The linker treats them separately.

Program Section Order

The linker determines the memory allocation of p-sects by the order of occurrence of the p-sects in the input modules. Table 11–3 shows the order in which p-sects appear for both overlaid and nonoverlaid files.

Table 11–3: P-sect Order

Nonoverlaid	Overlaid
absolute (. ABS) blank named (NAME)	absolute (. ABS) overlay handler ($OHAND) overlay table ($OTABL) blank named (NAME)

If there is more than one named section, the named sections appear in the order in which they occur in the input files. For example, the FORTRAN compiler arranges the p-sects in the main program module so that the USR can swap over pure code in low memory rather than over data required by the function making the USR call.

If the size of the blank p-sect is 0, it does not appear in the load map.

11.1.3 Global Symbols: Communication Links Between Modules

Global symbols provide the link, or communication, between object modules. You create global symbols with the .GLOBL or .ENABL GBL assembler directive (or with double colon, ::, double equals sign, = =, or by = =:). If the global symbol is defined in an object module (as a label using :: or by direct assignment using = =), other object modules can reference it. If the global

symbol is not defined in the object module, it is an external symbol and is assumed to be defined in some other object module. If a global symbol is used as a label in a routine, it is often called an entry point — that is, it is an entry point to that subroutine.

As the linker reads the object modules it keeps track of all global symbol definitions and references. It then modifies the instructions and data that reference the global symbols. The linker always prints undefined globals on the console terminal after pass 1. If you request a load map on the terminal, they also appear at the end of the load map.

Table 11–4 shows how the linker resolves global references when it creates the load module.

Table 11–4: Global Reference Resolution

Module Name	Global Definition	Global Reference
IN1	B1 B2	A L1 C1 XXX
IN2	A B1	B2
IN3		B1

In processing the first module, IN1, the linker finds definitions for B1 and B2, and references to A, L1, C1, and XXX. Because no definition currently exists for these references, the linker defers the resolution of these global symbols. In processing the next module, IN2, the linker finds a definition for A that resolves the previous reference, and a reference to B2 that can be immediately resolved.

When all the object modules have been processed, the linker has three unresolved global references remaining: L1, C1, and XXX. A search of the default system library resolves XXX. The global symbols L1 and C1 remain unresolved and are, therefore, listed as undefined global symbols.

The relocatable global symbol, B1, is defined twice and is listed on the terminal as a global symbol with multiple definitions. The linker uses the first definition of such a symbol. An absolute global symbol can be defined more than once without being listed as having multiple definitions, as long as each occurrence of the symbol has the same value.

11.2 Calling and Using the Linker

To call the linker from the system device, respond to the dot printed by the keyboard monitor by typing:

```
R LINK <RET>
```

The Command String Interpreter prints an asterisk at the left margin of the console terminal when it is ready to accept a command line. If you enter only a carriage return at this point, the linker prints its current version number.

Type two CTRL/Cs to halt the linker at any time (or a single CTRL/C to halt the linker when it is waiting for console terminal input) and return control to the monitor. To restart the linker, type R LINK or REENTER in response to the monitor's dot.

The first command string you enter in response to the linker's prompt has this syntax:

```
[bin-filespec],[map-filespec],[stb-filespec] = obj-filespec[/option...][,...obj-filespec[/option...]]
```

where:

bin-filespec	represents the device, file name, and file type to be assigned to the linker's output load module file
map-filespec	represents the device, file name, and file type of the load map output file
stb-filespec	represents the device, file name, and file type of the symbol definition file
obj-filespec	represents an object module, a library file, or a symbol table file, created in a previous link
/option	is one of the options listed in Table 11–6

In each file specification above, the device should be a random-access device, with these exceptions: the output device for the load map file can be any RT–11 device, as can the output device for an .LDA file if you use the /L option. If you do not specify a device, the linker uses default device DK:. Note that the linker load map contains lower-case characters. Use the SET LP LC command to enable lower-case printing if your printer has lower-case characters.

If you do not specify an output file, the linker assumes that you do not want the associated output. For example, if you do not specify the load module and load map (by using a comma in place of each file specification) the linker prints only error messages, if any occur.

Table 11–5 shows the default values for each specification.

Table 11–5: Linker Defaults

	Device	File Name	File Type
Load Module	DK:	none	SAV, REL(/R), LDA(/L)
Map Output	DK: or same as load module	none	MAP

(continued on next page)

Table 11–5: Linker Defaults (Cont.)

	Device	File Name	File Type
Symbol Definition Output	DK: or same as previous output device	none	STB
Object Module	DK: or same as previous object module	none	OBJ

If you make a syntax error in a command string, the system prints an error message. You can then retype the new command string following the asterisk. Similarly, If you specify a nonexistent file, a warning error occurs; control returns to the Command String Interpreter, an asterisk prints and you can reenter the command string.

Table 11–6 lists the options associated with the linker. You must precede the letter representing each option by the slash character. Options must appear on the line indicated if you continue the input on more than one line, but you can position them anywhere on the line. The column titled *Command Line* lists on which line in the command string the option can appear. (Section 11.5 provides a more detailed explanation of each option.)

Table 11–6: Linker Options

Option Name	Command Line	Section	Explanation
/A	first	11.5.1	Lists global symbols in program sections in alphabetical order.
/B:n	first	11.5.2	Changes the bottom address of a program to *n* (invalid with /H and /R).
/C	any but last	11.5.3	Continues input specification on another command line. (You can also use /C /V with /O; do not use /C with the // option.)
/E:n	first	11.5.4	Extends a particular program section in the root to a specific value.
/F	first	11.5.5	Instructs the linker to use the default FORTRAN library, FORLIB.OBJ; this option is provided only for compatibility with previous versions of RT–11.
/G	first	11.5.6	Adjusts the size of the linker's library directory buffer to accommodate the largest multiple definition library directory.
/H:n	first	11.5.7	Specifies the top (highest) address to be used by the relocatable code in the load module. Invalid with /B, /R, /Y and /Q.
/I	first	11.5.8	Extracts the global symbols you specify (and their associated object modules) from the library and links them into the load module.

(continued on next page)

Table 11–6: Linker Options (Cont.)

Option Name	Command Line	Section	Explanation
/K:n	first	11.5.9	Inserts the value you specify (the valid range for *n* is from 1 to 28) into word 56 of block 0 of the image file; this option is provided only for compatibility with the RSTS operating system. Invalid with /R.
/L	first	11.5.10	Produces a formatted binary output file (invalid for overlaid programs and for foreground links).
/M[:n]	first	11.5.11	Cause the linker to prompt you for a global symbol that represents the stack address or that sets the stack address to the value *n*. Do not use with /R.
/O:n	any, but first	11.5.12	Indicates that the program is an overlay structure; *n* specifies the overlay region to which the module is assigned. Invalid with /L.
/P:n	first	11.5.13	Changes the default amount of space the linker uses for a library routines list.
/Q	first	11.5.14	Lets you specify the base addresses of up to eight root program sections. Invalid with /H or /R.
/R[:n]	first	11.5.15	Produces output in relocatable format and can indicate stack size for a foreground job. Invalid with /B, /H, /K, and /L, /V.
/S	first	11.5.16	Makes the maximum amount of space in memory available for the linker's symbol table. (Use this option only when a particular link stream causes a symbol table overflow.)
/T[:n]	first	11.5.17	Cause the linker to prompt you for a global symbol that represents the transfer address or that sets the transfer address to the value *n*.
/U:n	first	11.5.18	Rounds up the root program section you specify so that the size of the root segment is a whole number multiple of the value you supply (*n* must be a power of 2).
/V	first	11.5.19	Enables special .SETTOP and .LIMIT features provided by the XM monitor. Invalid with /R and /L.
/V:n[:m]	any but first	11.5.19	Indicates that an extended memory overlay segment is to be mapped in virtual region *n*, and optionally in partition *m*.
/W	first	11.5.20	Directs the linker to produce a wide load map listing.
/X	first	11.5.21	Does not output the bitmap if the code is placed over the bitmap (location 360–377). This option is provided only for compatibility with the RSTS operating system.
/Y:n	first	11.5.22	Starts a specific program section in the root on a particular address boundary. Invalid with /H.
/Z:n	first	11.5.23	Sets unused locations in the load module to the value *n*.
//	first and last	11.5.3	Allows you to specify command string input on additional lines. Do not use this option with /C.

11.3 Input and Output

Linker input and output is in the form of modules; the linker uses one or more input modules to produce a single output (load) module. The linker also accepts library modules and symbol table definition files as input, and can produce a load map and/or symbol table definition file. The sections that follow describe all valid forms of input to and output from the linker.

11.3.1 Input Object Modules

Object files, consisting of one or more object modules, are the input to the linker. (Entering files that are not object modules may result in a fatal error.) Object modules are created by language translators such as the FORTRAN compiler and the MACRO–11 assembler. The module name item declares the name of the object module (see Section 11.3.4).

The first six Radix–50 characters of the .TITLE assembler directive are used as the name of the object module. These six characters must be Radix–50 characters (the linker ignores any characters beyond the sixth character). The linker prints the first module name it encounters in the input file stream (normally the main routine of the program) on the second line of the map following TITLE:. The linker also uses the first identity label (issued by the .IDENT directive) for the load map. It ignores additional module names.

The linker reads each object module twice. During the first pass it reads each object module to construct a global symbol table and to assign absolute values to the program section names and global symbols. The linker uses the library files to resolve undefined globals. It places their associated object modules in the root. On the second of its two passes, the linker reads the object modules, links and relocates the modules, and outputs the load module.

Symbol table definition files are special object files that can serve as input to LINK anywhere other object files are allowed.

11.3.2 Input Library Modules

The RT–11 linker can automatically search libraries. Libraries consist of library files, which are specially formatted files produced by the librarian program (described in Chapter 12) that contain one or more object modules. The object modules provide routines and functions to aid you in meeting specific programming needs. (For example, FORTRAN has a set of modules containing all necessary computational functions — SQRT, SIN, COS, and so on.) You can use the librarian to create and update libraries. Then you can easily access routines that you use repeatedly or routines that different programs use. Selected modules from the appropriate library file are linked as needed with your program to produce one load module. Libraries are further described in Chapter 12.

NOTE

Library files that you combine with the monitor COPY command or with the PIP /U or /B option are invalid as input to both the linker and the librarian.

You specify libraries in a command string in the same way as you specify normal modules; you can include them anywhere in the command string. If you are creating an overlay structure, specify libraries before you specify the overlay structure. Do not specify libraries on the same line as overlay segments. If a global symbol is undefined at the time the linker encounters the library in the input stream, and if a module is included in the library that contains that global definition, then the linker pulls that module from the library and links it into the load image. Only the modules needed to resolve references are pulled from the library; unreferenced modules are not linked.

Modules in one library can call modules from another library; however, the libraries must appear in the command string in the order in which they are called. For example, assume module X in library ALIB calls Y from the BLIB library. To correctly resolve all globals, the order of ALIB and BLIB should appear in the command line as:

```
*Z=B,ALIB,BLIB
```

Module B is the root. It calls X from ALIB and brings X into the root. X in turn calls Y, which is brought from BLIB into the root.

Library Module Processing

The linker selectively relocates and links object modules from specific user libraries that were built by the librarian. Figure 11–1 diagrams this general process. During pass 1 the linker processes the input files in the order in which they appear in the input command line. If the linker encounters a library file during pass 1, it takes note of the library in an internal save status block, and then proceeds to the next file. The linker processes only non-library files during the initial phase of pass 1. In the final phase of pass 1 the linker processes only library files. This is when it resolves the undefined globals that were referenced by the non-library files.

The linker processes library files in the order in which they appear in the input command line. The default system library (SY:SYSLIB.OBJ) is always processed last.

The search method the linker uses allows modules to appear in any order in the library. You can specify any number of libraries in a link and they can be positioned anywhere, with the exception of forward references between libraries, and they must come before the overlay structure. The default system library, SY:SYSLIB.OBJ, is the last library file the linker searches to resolve any remaining undefined globals.

Some languages, such as FORTRAN, have an Object Time System (OTS) that the linker takes from a library and includes in the final module. The most efficient way to accomplish this is to include these OTS routines (such

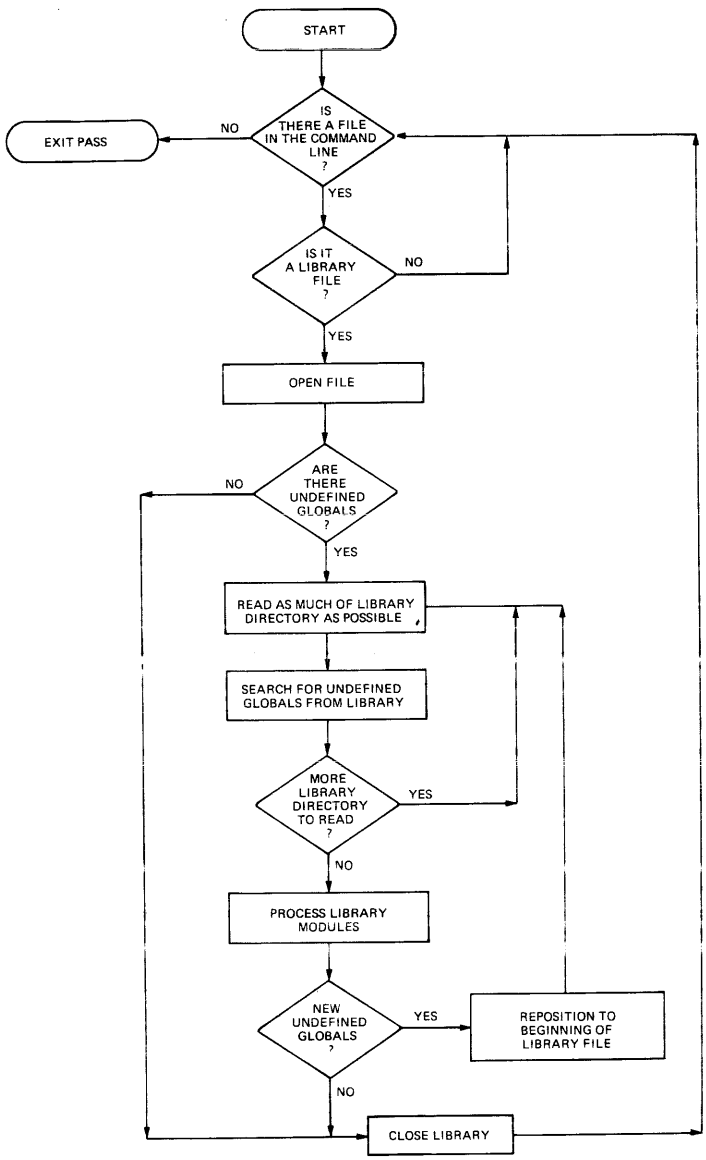
as NHD, OTSCOM, and V2NS for FORTRAN) in SY:SYSLIB.OBJ. See the *RT–11 Installation and System Generation Guide* for details on how to do this.

Libraries are input to the linker the same way as other input files. Here is a sample LINK command string:

```
*TASK01,LP:=MAIN,MEASUR
```

This causes program MAIN.OBJ to be read from DK: as the first input file. Any undefined symbols generated by program MAIN.OBJ should be satisfied by the library file MEASUR.OBJ specified in the second input file. The linker tries to satisfy any remaining undefined globals from the default library, SY:SYSLIB.OBJ. The load module, TASK01.SAV, is stored on DK: and a load map prints on the line printer.

Figure 11–1: Library Searches



Multiple Definition Libraries

In addition to the libraries explained so far, LINK processes multiple definition libraries. DIGITAL does not recommend that you use this type of library in normal situations; its primary purpose is to provide special functions for RSTS. These libraries differ from other libraries in that they can contain more than one definition for a given global. You specify multiple definition libraries in the command line the same way you specify normal libraries. Modules that LINK obtains from multiple definition libraries always appear in the root.

It is useful to be aware of the differences between processing normal and multiple definition libraries. When you include modules from a multiple definition library, LINK has to store that library's directory in an internal buffer. A library's directory is often called an entry point table (EPT). If a library EPT is too large to fit into the internal buffer, LINK prints a message instructing you to use the /G option. the /G option changes the buffer's size to accommodate the largest EPT of all the multiple definition libraries you are using. Use the /G option only when LINK indicates it is required.

When a global symbol from a multiple definition library matches an undefined global, LINK removes from the undefined global list all other globals defined in the same library. LINK does this before it processes the library module. Thus, two modules with identical globals will not appear in the linked module.

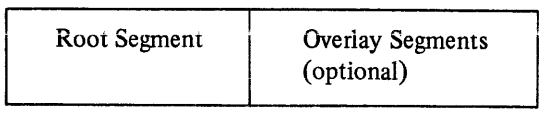
NOTE

The order of modules in multiple definition libraries is very important and will affect which modules LINK uses. The increased EPT size (due to duplicate entries, in addition to module name entries) will also slow LINK down.

11.3.3 Output Load Module

The primary output of the linker is a load module that you can run under RT–11. The linker creates as a load module a memory image file (file type of .SAV) for use under a single-job system or the background job, and save images can also be run as virtual foreground jobs in the XM monitor. If you need to execute a program in the foreground, use the /R option to produce a relocatable format (file type of .REL) foreground load module. The linker can produce an absolute load module (file type of .LDA) if you need to load the module with the Absolute Loader. See the *RT–11 Software Support Manual* for more details on these formats.

The load module for a memory image file is arranged as follows:



For a relocatable image file the load modules are arranged as follows:

Root Segment	Overlay Segments (optional)	Relocation information for root and overlay segments

The first 256-word block of the root segment (main program) contains the memory usage bitmap and the locations the linker uses to pass program control parameters. The memory usage bitmap outlines the blocks of memory that the load module uses; it is located in locations 360 through 377.

Table 11–7 lists the parameters that appear in the absolute block, the addresses the parameters occupy, and the conditions under which they are set.

Table 11–7: Absolute Block Parameters Information

Address	Information	When Set
0	Identification of a program that was created with /V option	only with /V
2	Highest virtual memory address used by the program	only with /V
14,16	(XM only) BPT trap	only with /R
20,22	(XM only) IOT trap	only with /R
34,36	TRAP vector	only with /R
40	Start address of program	always
42	Initial setting of SP (stack pointer)	always
44	Job Status Word (overlay bit set by LINK)	always
46	USR swap (set by user) address (0 implies normal location)	always
50	Highest memory address used by the program (high limit)	always
52	Size of root segment in bytes	only with /R
54	Stack size in bytes (value with /R or default 128)	only with /R
56	Size of overlay region in bytes	only with /R
60	Identification of a file in relocatable (REL) format	only with /R
62	Relative block number for start of relocation information	only with /R
64	Start address of overlay table	with /O or /V
66	Start of the virtual overlay segment information in overlay handler tables	only with /V
360–377	Memory usage bitmap	always, except with /X or /L

The linker stores default values in locations 40, 42, and 50, unless you use options to specify otherwise. The /T option affects location 40, for example, and /M affects location 42. You can also use the .ASECT directive to change the defaults. The overlay bit is located in the job status word. LINK auto-

matically sets this bit if the program is overlaid. Otherwise, the linker initially sets location 44 to 0. Location 46 also contains zero unless you specify another value by using the .ASECT directive.

You can assign initial values to memory locations 0–476 (which include the interrupt vectors and system communication area) by using an .ASECT assembler directive. The values appear in block 0 of the load module, but there are restrictions on the use of .ASECT directives in this region. You should not modify location 54 or locations 360–377 because the memory usage map is passed in those locations. In addition, for foreground links, modifications of words 52–62 are not permitted because additional parameters are passed to the FRUN command in those locations.

You can use an ASECT directive to set any location that is not restricted, but be careful if you change the system communication area. The program itself must initialize restricted areas, such as locations 360–377. There are no restrictions on .ASECT directives if the output format is LDA.

Locations in addresses 0–476 might not be loaded at execution time, even though your program uses an ASECT to initialize them. For background programs, this is because the R, RUN, and GET commands do not load addresses that are protected by the monitor's memory protection map. For foreground programs, the FRUN command loads only locations 14–22 and 34–50 and ignores all other low memory location. To initialize a location at run time, use the .PROTECT programmed request. If it is successful, follow it with a MOV instruction to modify the location.

11.3.4 Output Load Map

The linker can produce a load map following the completion of the initial pass. This map, shown in Figure 11–2, diagrams the layout of memory for the load module.

The load map lists each program section that is included in the linking process. The line for a section includes the name and low address of the section and its size in bytes. The rest of the line lists the program section attributes, as shown in Table 11–3. The remaining columns contain the global symbols found in the section and their values.

The map begins with the linker version number, followed by the date and time the program was linked. The second line lists the file name of the program, its title (which is determined by the first module name record in the input file), and the first identification record found. The absolute section is always shown first, followed by any nonrelocatable symbols. The modules located in the root segment of the load module are listed next, followed by those modules that were assigned to overlays in order by their region number (see Section 11.6). Any undefined global symbols are then listed. The map ends with the transfer address (start address) and high limit of relocatable code in both octal bytes and decimal words.

NOTE

The load map does not reflect the absolute addresses for a REL file that you create to run as a foreground job; you must add the base relocation address determined at FRUN time to obtain the absolute addresses. The linker assumes a base address of 1000.

For example, assume the FRUN command is used to run the program TEST:

```
.FRUN TEST/P
Loaded at 127276
```

The /P option causes FRUN to print the load address, which is 127276 in this example. To calculate the actual location in memory of any global in the program, first subtract 1000 from that global's value. (The value 1000 represents the base address assigned by the linker. This offset is not used at load time.) Then add the result to the load address determined with /P. The final result represents the absolute location of the global.

Figure 11–2: Sample Load Map

```
1  RT-11 LINK V06.01       Load Map          Wed 20-Jun-79 11:25:28
2  TEST  .SAV    Title:    TEST     Ident:
3
4  Section  Addr   Size     Global  Value   Global  Value   Global  Value
5
6    . ABS. 000000 001000   (RW,I,GBL,ABS,OVR)
7           001000 000200   (RW,I,LCL,REL,CON)
8    TEST   001200 000174   (RW,I,LCL,REL,CON)
9                           START   001200  EXIT    001240
10
11 Transfer address = 001200, High limit = 001372 = 381.  words
```

Table 11–8 describes each line in the sample load map above.

Table 11–8: Line-by-Line Sample Load Map Description

Line	Contents
1	Load map header
2	Program name, program title (.MAIN. default) and identity (default is blank)
4	P-sect description header. *Section* indicates the p-sect name; *Addr* indicates the p-sect start address; *Size* indicates p-sect length in octal bytes; *Global* and *Value* list the p-sect globals and their associated octal values.
6	Absolute p-sect, . ABS. This line includes the absolute p-sect's start address, length and attributes (for a complete description of these abbreviations, see Table 11–3). The linker always includes a . ABS. p-sect in the link.
7	Unnamed p-sect. This p-sect appears in the load map after the absolute p-sect. For overlaid programs, the unnamed p-sect appears in the load map after the overlay table p-sect (see Figure 11–11).
8–9	TEST p-sect. Line 9 lists TEST's two globals, START and EXIT, with their associated values.
11	*Transfer address* indicates the address in memory where the program starts. *High limit* indicates the last address used by the program. The number of words in the program appears last.

11.4 Creating an Overlay Structure

The ability of RT–11 to handle overlays gives you virtually unlimited memory space for an assembly language or FORTRAN program. A program using overlays can be much larger than would normally fit in the available memory space, since portions of the program reside on a storage device such as disk or DECtape. To utilize this capability, you must define an overlay structure for your program.

Prior to version 4, RT–11 permitted overlays to be placed only in low memory. Now you can place them in extended memory, too, if you run your program on a system that has an extended memory configuration and XM monitor. Overlays that reside in low memory are called low memory overlays, and those in extended memory are called extended memory overlays.

Section 11.4.1, Low Memory Overlays, describes low memory overlays in general and shows how to define a low memory overlay structure for your program. Section 11.4.2, Extended Memory Overlays, deals specifically with extended memory overlays, and shows how to define an overlay structure that has either extended memory overlays only or both extended memory and low memory overlays.

Read 11.4.1 before reading 11.4.2, because much of the information contained in the first subsection applies to the second subsection.

11.4.1 Low Memory Overlays

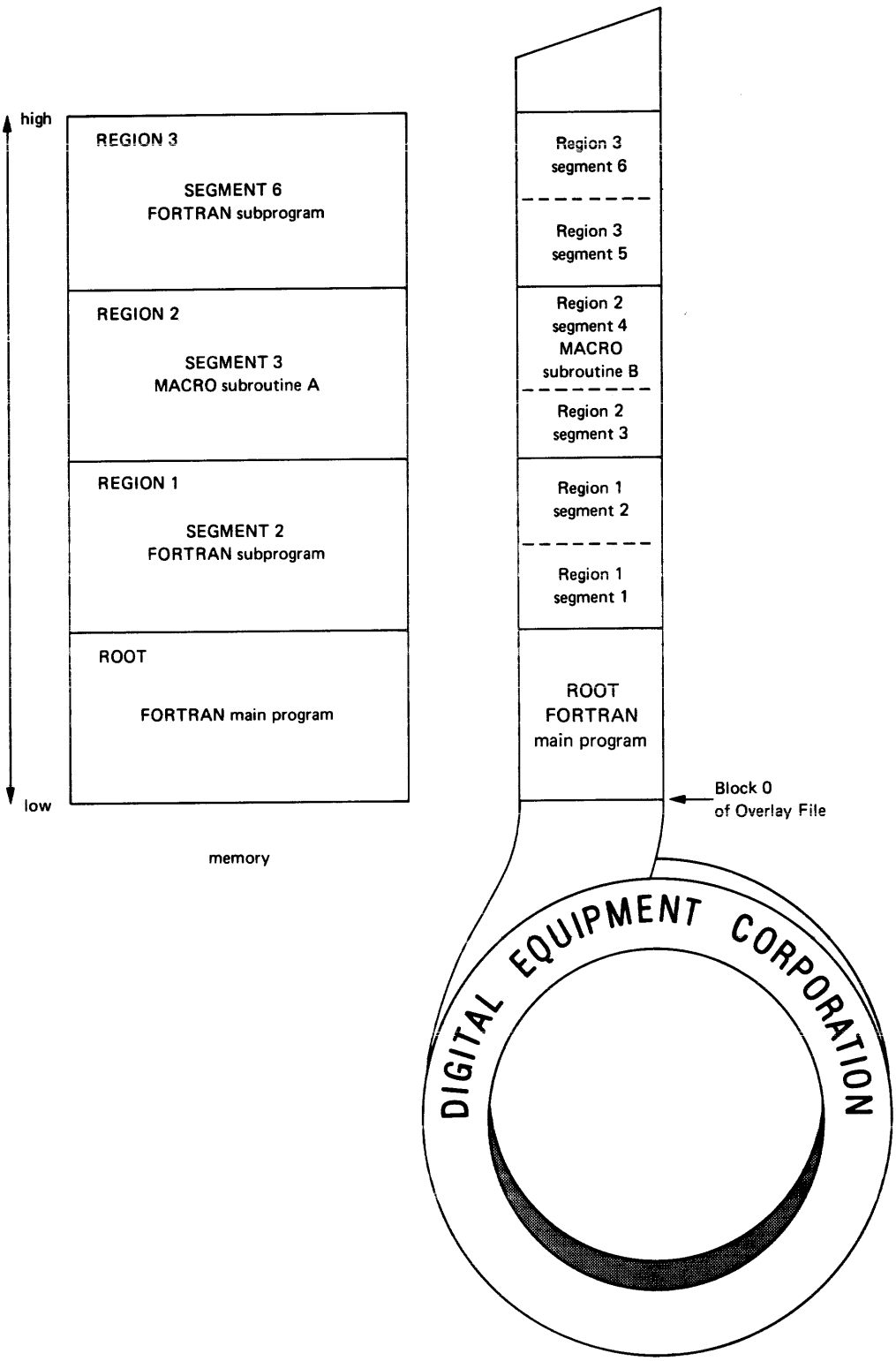
An overlay structure divides a program into segments. For each overlaid program there is one root segment and a number of overlay segments. Each overlay segment is assigned to a particular area of available memory called an overlay region. More than one overlay segment can be assigned to a given overlay region. However, each region of memory is occupied by one (and only one) of its assigned segments at a time. The other segments assigned to that region are stored on disk, diskette, DECtape, or DECtape II. They are brought into memory when called, replacing (overlaying) the segment previously stored in that region. The root segment, on the other hand, contains those parts of the program that must always be memory resident. Therefore the root is never overlaid by another segment.

Figure 11–3 diagrams an overlay structure for a FORTRAN program. The main program is placed in the root segment and is never overlaid. The various MACRO subroutines and FORTRAN subprograms are placed in overlay segments. Each overlay segment is assigned to an overlay region and stored on DECtape until called into memory. For example, region 2 is shared by the MACRO subroutine A currently in memory and the MACRO subroutine B in segment 4. When a call is made to subroutine B, segment 4 is brought into region 2 of memory, overlaying or replacing segment 3.

The overlay file, shown on the DECtape in Figure 11–3, is created by the linker when you specify an overlay structure. The overlay file contains at all

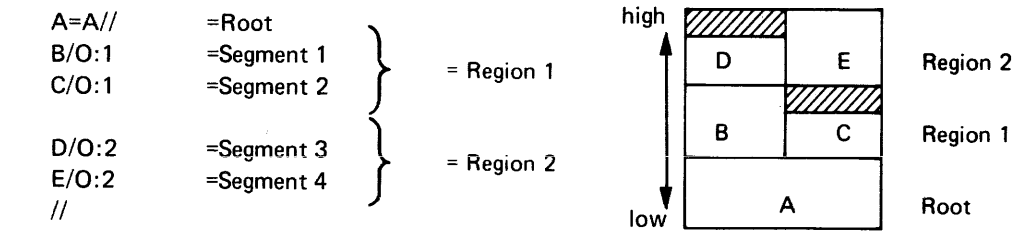
times a copy of the root segment and each overlay segment, including those overlay segments currently in memory.

Figure 11–3: Sample Overlay Structure for a FORTRAN Program



You specify an overlay structure to the linker by using the /O option (see Figure 11–4). To specify an overlay structure that uses extended memory, use the /V option (see Section 11.4.2 for a discussion of extended memory overlays). This option is described fully in Section 11.4.2.4.

Figure 11–4: Overlay Scheme



The linker calculates the size of any region to be the size of the largest segment assigned to that region. Thus, to reduce the size of a program (that is, the amount of memory it needs), you should first concentrate on reducing the size of the largest segment in each region. The linker delineates the overlay regions you specify, and prefaces your program with the run-time overlay handler code shown in Figure 11–5. The linker also sets up links between the overlay handler and program references to routines that reside in overlays. When, at run time, a reference is made to a section of your program that is not currently in memory, these links cause an overlay to be read into memory. The overlay segment containing the referenced code becomes resident.

Figure 11–5: The Run-Time Overlay Handler

```
.SBTTL  THE RUN-TIME OVERLAY HANDLER

;+
; THE FOLLOWING CODE IS INCLUDED IN THE USER'S PROGRAM BY THE
; LINKER WHENEVER LOW MEMORY OVERLAYS ARE REQUESTED BY THE USER.
; THE RUN-TIME LOW MEMORY OVERLAY HANDLER IS CALLED BY A DUMMY
; SUBROUTINE OF THE FOLLOWING FORM:
;
;       JSR     R5,$OVRH        ;CALL TO COMMON CODE FOR LOW MEMORY OVERLAYS
;       .WORD   <OVERLAY # *6>  ;# OF DESIRED SEGMENT
;       .WORD   <ENTRY ADDR>    ;ACTUAL CORE ADDR (VIRTUAL ADDR)
;
; ONE DUMMY ROUTINE OF THE ABOVE FORM IS STORED IN THE RESIDENT PORTION
; OF THE USER'S PROGRAM FOR EACH ENTRY POINT TO A LOW MEMORY OVERLAY SEGMENT.
; ALL REFERENCES TO THE ENTRY POINT ARE MODIFIED BY THE LINKER TO BE
; REFERENCES TO THE APPROPRIATE DUMMY ROUTINE.  EACH OVERLAY SEGMENT
; IS CALLED INTO CORE AS A UNIT AND MUST BE CONTIGUOUS IN CORE.  AN
; OVERLAY SEGMENT MAY HAVE ANY NUMBER OF ENTRY POINTS, TO THE LIMITS
; OF CORE MEMORY.  ONLY ONE SEGMENT AT A TIME MAY OCCUPY AN OVERLAY REGION.
;
; THERE IS ONE WORD PREFIXED TO EVERY OVERLAY REGION THAT IDENTIFIES THE
; SEGMENT CURRENTLY RESIDENT IN THAT OVERLAY REGION.  THIS WORD IS AN INDEX
; INTO THE OVERLAY TABLE, AND POINTS AT THE OVERLAY SEGMENT INFORMATION.
;
; UNDEFINED GLOBALS IN THE OVERLAY HANDLER MUST BE NAMED "$OVDF1" TO
; "$OVDFn" SUCH THAT A RANGE CHECK MAY BE DONE BY LINK TO DETERMINE IF
; THE UNDEFINED GLOBAL NAME IS FROM THE OVERLAY HANDLER. A CHECK IS
; DONE ON THE .RAD50 CHARACTERS "$OV", AND THEN A RANGE CHECK IS DONE ON
; THE .RAD50 CHARATERS "DF1" TO "DFn". THESE GLOBAL SYMBOLS DO NOT APPEAR
```

(continued on next page)

```
; ON LINK MAPS, SINCE THEIR VALUE IS NOT KNOWN UNTILL AFTER THE MAP HAS BEEN
; PRINTED. CURRENTLY $OVDF1 TO $OVDF5 ARE IN USE.
;
; GLOBAL SYMBOLS O$READ, AND O$DONE ARE USEFULL WHEN DEBUGGING
; OVERLAID PROGRAMS.
;
;       O$READ:: WILL APPEAR IN THE LINK MAP, AND LOCATES THE .READW
;       STATEMENT IN THE OVERLAY HANDLER.
;
;       O$DONE:: WILL APPEAR IN THE LINK MAP, AND LOCATES THE FIRST
;       INSTRUCTION AFTER THE .READW IN THE OVERLAY HANDLER.
;-

.MCALL  .READW,,,V1,,
        ,,V1,,          ;V1 FORMAT

.SBTTL  OVERLAY HANDLER CODE

.PSECT  $OHAND,GBL

.ENABL  GBL
.ENABL  LSB

; $OVRH IS THE ENTRY POINT TO THE OVERLAY HANDLER

        .RAD50  /OVR/           ;THIS KEEPS HANDLER THE SAME SIZE AS V03
$OVRH:: MOV     R0,-(SP)        ;/O OVERLAY ENTRY POINT
        MOV     R1,-(SP)        ;SAVE REGISTERS
        MOV     R2,-(SP)

1$:
        BR      5$              ;FIRST CALL ONLY * * *
;       MOV     @R5,R1          ;PICK UP OVERLAY NUMBER
        ADD     #$OVTAB-6,R1    ;CALC TABLE ADDR
        MOV     (R1)+,R2        ;GET FIRST ARG. OF OVERLAY SEG. ENTRY
2$:     CMP     (R5)+,@R2       ;IS OVERLAY ALREADY RESIDENT?
        BEQ     3$              ;YES, BRANCH TO IT

;+
; THE .READW ARGUMENTS ARE AS FOLLOWS:
; CHANNEL NUMBER, CORE ADDRESS, LENGTH TO READ, RELATIVE BLOCK ON DISK.
; THESE ARE USED IN REVERSE ORDER FROM THAT SPECIFIED IN THE CALL.
;-

O$READ::.READW  17,R2,@R1,(R1)+ ;READ FROM OVERLAY FILE
O$DONE::BCS     4$
3$:     MOV     (SP)+,R2        ;RESTORE USERS REGISTERS
        MOV     (SP)+,R1
        MOV     (SP)+,R0
        MOV     @R5,R5          ;GET ENTRY ADDRESS
        RTS     R5              ;ENTER OVERLAY ROUTINE AND RESTORE USER'S R5

4$:     EMT     376             ;SYSTEM ERROR 10 (OVERLAY I/O)
        .BYTE   0,373

5$:     MOV     #11501,1$       ;RESTORE SWITCH INSTR (MOV @R5,R1)
        MOV     $ODF1,R1        ;START ADDR FOR CLEAR OPERATION
6$:     CLR     (R1)+           ;CLEAR ALL OVERLAY REGIONS
        CMP     R1,$ODF2        ;DONE?
        BLO     6$              ;LO -> NO, REPEAT
        BR      1$              ;AND RETURN TO CALL IN PROGRESS

$ODF1:: .WORD   $OVDF1          ;HIGH ADDR OF ROOT SEGMENT + 2 (NXT AVAIL)
$ODF2:: .WORD   $OVDF2          ;HIGH ADDRESS OF /O OVERLAYS +2 (NXT AVAIL)

.DSABL  LSB

.SBTTL  $OVTAB  OVERLAY TABLE

;+
; OVERLAY TABLE STRUCTURE:
;
; LOC 64 ->     $OVTAB:
;               .WORD   <CORE ADDR>,<RELATIVE BLK>,<WORD COUNT>   /O OVERLAYS
;               DUMMY SUBROUTINES FOR ALL OVERLAY SEGMENTS
;-
```

(continued on next page)

```
.PSECT  $OTABL,D,GBL,OVR
$OVTAB:
.END
```

There is no special formula for creating an overlay structure. You do not need a special code or function call. However, some general guidelines must be followed. For example, a FORTRAN main program must always be placed in the root segment. This is true also for a global program section (such as a named COMMON block) that is referenced by more than one overlay segment.

The assignment of region numbers to overlay segments is crucial. Segments that overlay each other (have the same region number) must be logically independent; that is, the components of one segment cannot reference the components of another segment assigned to the same region. Segments that need to be memory resident simultaneously must be assigned to different regions.

When you make calls to routines or subprograms that are in overlay segments, the entire return path must be in memory. This means that from an overlay segment you cannot call a routine that is in a different segment of the same region. If this is done, the called routine overlays the segment making the call, and destroys the return path.

Figure 11–6 illustrates a sample set of subroutine calls and return paths. In the example, solid lines represent legal subroutine calls and dotted lines represent invalid calls.

Suppose the following subroutine calls were made:

1. The root calls segment 8
2. 8 calls segment 4
3. Segment 4 calls segment 3

Segment 3 can now call any of the following, in any order:

itself	segment 8
segment 4	the root

These segments and the root, of course, are all currently resident in memory.

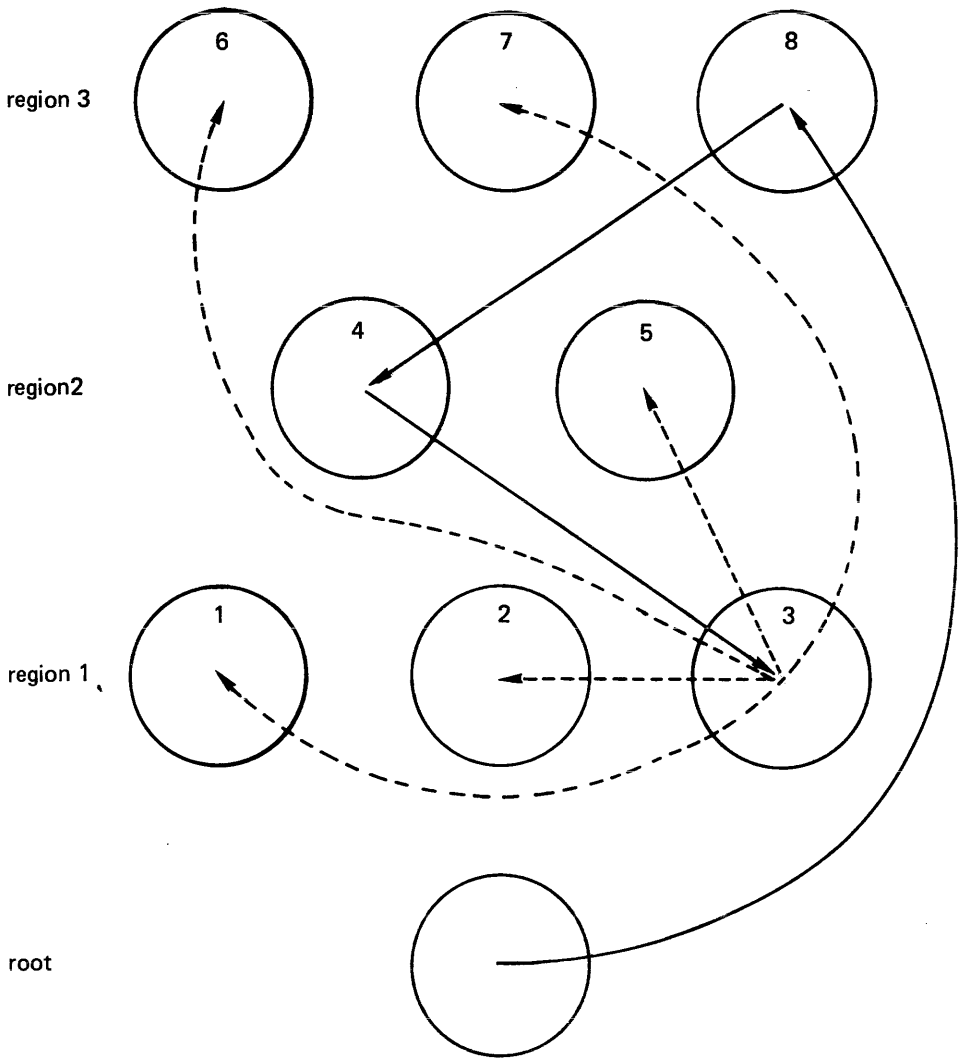
Segment 3 cannot call any of the following segments because this would destroy its return path:

segments 2 and 1
segment 5
segments 6 and 7

Look at what might happen if one of these invalid calls is made. Assume that segments 3, 4, and 5 all contain MACRO subroutines. Suppose segment 4 calls segment 3 and segment 3 in turn calls segment 5. Segment 5 is not

resident in region 2, so an overlay read-in occurs: segment 5 is read into memory, thus destroying the memory-resident copy of segment 4. The subroutine in segment 5 executes and returns control to segment 3. Segment 3 finishes its task and tries to return control to segment 4. Segment 4, however, has been replaced in memory by segment 5. Segment 4 cannot regain control and the program loops indefinitely, or traps, or random results occur.

Figure 11–6: Sample Subroutine Calls and Return Paths



The guidelines already mentioned and some additional rules for creating overlay structures are summarized below.

1. SYSLIB must be present to create an overlay structure because it contains the overlay handler.
2. Overlay segments assigned to the same region must be logically independent; that is, the components of one segment cannot reference the components of another segment assigned to the same region.

3. The root segment contains the transfer address, stack space, impure variables, data, and variables needed by many different segments. The FORTRAN main program unit must be placed in the root segment.

4. A global program section (such as a named COMMON block or a .PSECT with the GBL attribute) that is referenced in more than one segment is placed in the root segment by the linker. This permits common access across the different segments.

5. Object modules that are automatically acquired from a library file will automatically be placed in an overlay segment, so long as that library module is referenced only by that segment. If a library module is referenced by more than one segment, LINK places that library module in the root.

   Do not specify a library file on the same command line as an overlay segment. You must specify all library modules before specifying any overlay modules. Link places in the root any modules from a multiple definition library and any modules included with the /I option.

6. All COMMON blocks that are initialized with DATA statements must be similarly initialized in the segment in which they are placed.

7. When you make calls to overlay segments, the entire return path to the calling routine must be in memory. (With XM overlays, the entire return path must be mapped. See Section 11.4.2.) This means you should take the following points into account:

   a. You can make calls with expected return (as from a FORTRAN main program to a FORTRAN or MACRO subroutine) from an overlay segment to entries in the same segment, the root segment, or to any other segment, so long as the called segment does not overlay in memory part of your return path to the main program.

   b. You can make jumps with no expected return (as in a MACRO program) from an overlay segment to any entry in the program with one exception: you can not make such a jump to a segment if the called segment will overlay an active routine (that is, a routine whose execution has begun, but not finished, and that will be returned to) in that region.

   c. Calls you make to entries in the same region as the calling routine must be entirely within the same segment, not within another segment in the same region.

8. You must make calls or jumps to overlay segments directly to global symbols defined in an instruction p-sect (entry points). For example, if ENTER is a global tag in an overlay segment, the first of the following two commands is valid, but the second is not:

```
JMP ENTER       ;VALID
JMP ENTER+6     ;INVALID
```

9. You can use globals defined in an instruction p-sect (entry points) of an overlay segment only for transfer of control and not for referencing data within an overlay segment. The assembler and linker cannot detect a violation of this rule so they issue no error. However, such a violation can cause the program to use incorrect data. If you reference these global symbols outside of their defining segment, the linker resolves them by using dummy subroutines of four words each in the overlay handler. Such a reference is indicated on the load map by a "@" following the symbol.

10. The linker directly resolves symbols that you define in a data p-sect. It is your responsibility to load the data into memory before referencing a global symbol defined in a data section.

11. You cannot use a section name to pass control to an overlay because it does not load the appropriate segment into memory. For example, JSR PC,OVSEC is illegal if you use OVSEC as a .CSECT name in an overlay. You must use a global symbol to pass control from one segment to the next.

12. In the linker command string, specify overlay regions in ascending order.

13. Overlay regions are read-only. Unlike USR swapping, an overlay handler does not save the segment it is overlaying. Any tables, variables, or instructions that are modified within a given overlay segment are reinitialized to their original values in the SAV or REL file if that segment has been overlaid by another segment. You should place any variables or tables whose values must be maintained across overlays in the root segment.

14. Your program cannot use channel 17 (octal) because overlays are read on that channel.

15. MACRO and FORTRAN directly resolve all global symbols that are defined in a module. If LINK moves the p-sect where they are defined from an overlay segment to the root, LINK will not generate an overlay table entry for those symbols.

Refer to the *RT–11/RSTS/E FORTRAN IV User's Guide* for additional information.

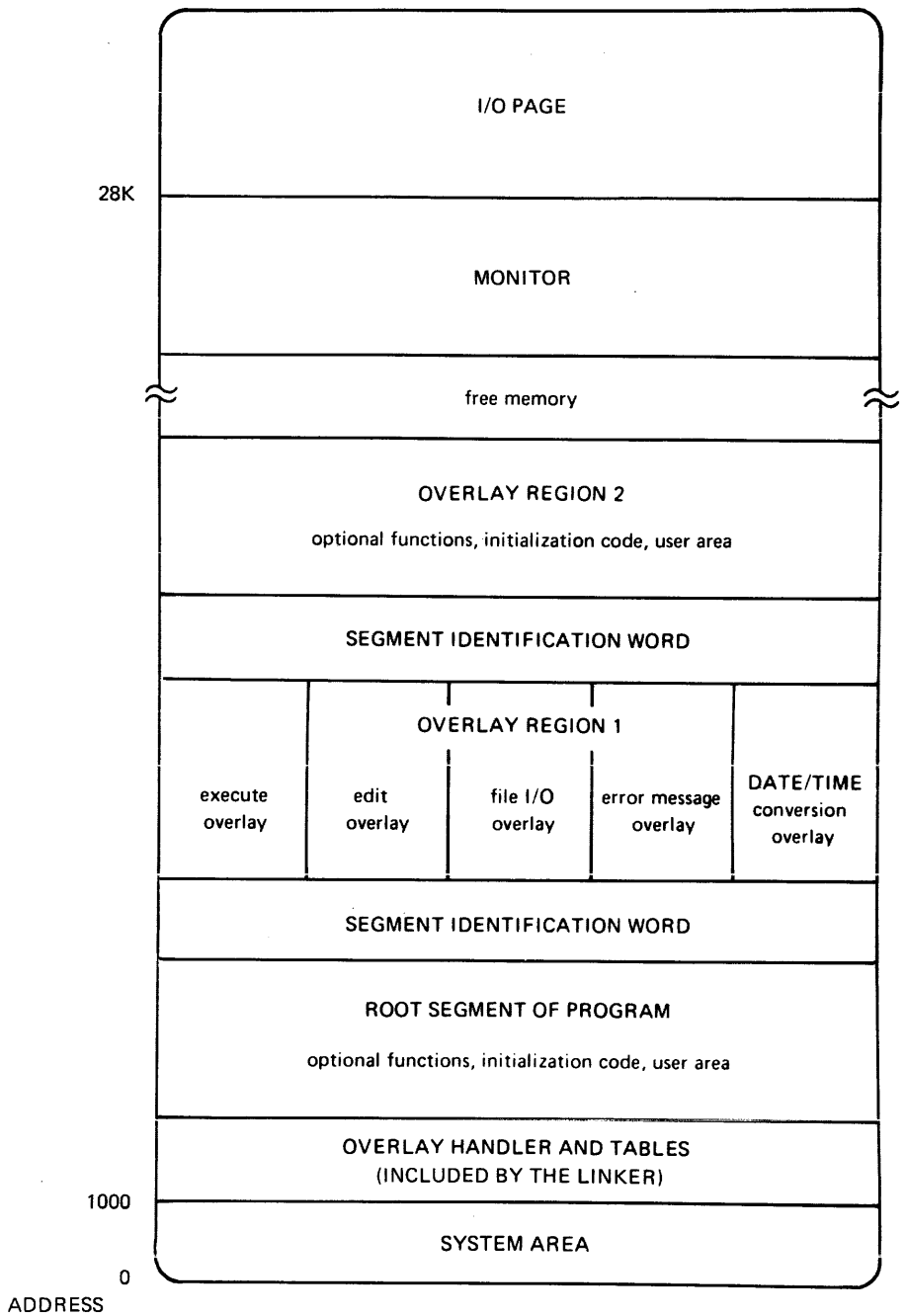
The absolute section (. ABS) never takes part in overlaying in any way. It is part of the root and is always resident.

This set of rules applies only to communications among the various modules that make up a program. Internally, each module must only observe standard programming rules for the PDP–11 (as described in the *PDP–11 Processor Handbook* and in the *FORTRAN* and *MACRO–11 Language Reference Manuals*).

Note that the condition codes set by your program are not preserved across overlay segment boundaries.

The linker provides overlay services by including a small resident overlay handler in the same file with your program to be used at program run time. The linker inserts this overlay handler plus some tables into your program beginning at the bottom address. The linker then moves your program up in memory to make room for the overlay handler and tables, if necessary. The handler is stored in SYSLIB. This scheme is diagrammed in Figure 11–7.

Figure 11–7: Memory Diagram Showing BASIC Link with Overlay Regions



11.4.2 Extended Memory Overlays

You can use LINK to create an overlay structure for your program that uses extended memory. Although you need a hardware configuration that includes a Memory Management Unit to run a program that has overlays in extended memory, you can link it on any RT–11 system. Read Section 11.4.1, Low Memory Overlays, before reading this section — much of the information contained in that section applies to extended memory overlays as well. Usually, you can convert an overlaid program to use extended memory without modifying the code. The extended memory overlay handler and the keyboard monitor include all the programmed requests necessary to access extended memory (see the *RT–11 Software Support Manual* for details on extended memory restrictions). The overlay tables also include additional data used by these requests, so you can access extended memory automatically without using extended memory programmed requests in your program. Refer to the *RT–11 Software Support Manual* for more information on extended memory.

The extended memory overlay structure is different from the low memory overlay structure in that extended memory overlays can reside concurrently in extended memory. This allows for speedier execution because, once read in, your program requires fewer I/O transfers with the auxiliary mass storage volume. If all program data is resident, and the program is loaded, the program may be able to run without an auxiliary mass storage volume. However, you must observe the same restrictions with extended memory overlays that apply to low memory overlays, especially regarding return paths. This section describes how to create a program with overlays in extended memory and ends with an example of such a program.

NOTE

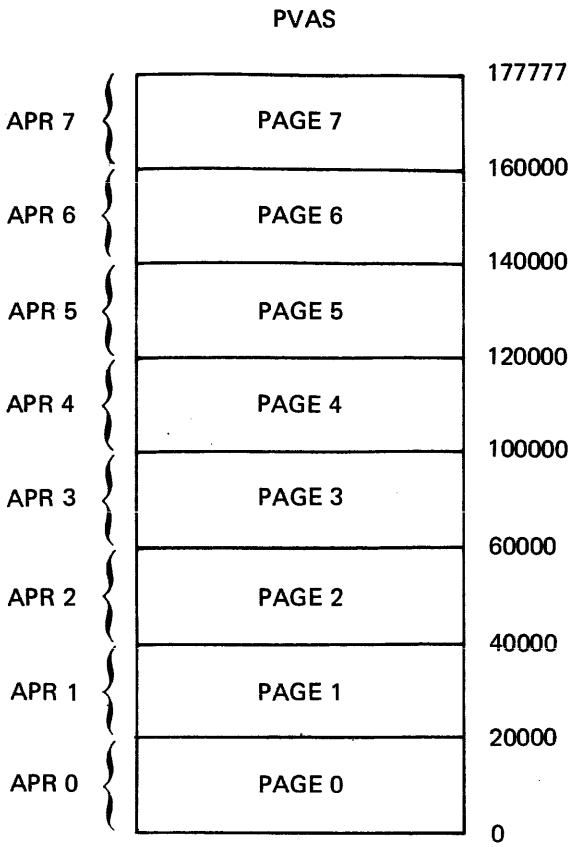
Overlays that reside in extended memory can contain impure data, but impure data is not automatically initialized each time a new overlay segment maps over a segment that contains impure data.

11.4.2.1 Virtual Address Space — When you set up an extended memory overlay structure, you set it up as though you had locations 0 to 177777 (that is, 32K words of memory) available for your use. Physically, not all these locations are available to you in low memory; your program's absolute section resides, typically, in locations 0 to 500, and the monitor takes up a good deal of memory starting at location 160000, going downward. Also, the computer sets aside addresses 160000 to 177777 for the I/O page. But, because of memory management, you can structure your program as though you had all 32K words of memory for your use. This space is called the program virtual address space (PVAS). The memory management hardware and the monitor will allow part of your 32K address space to reside in extended memory.

The program virtual address space is divided into eight sections called pages, numbered 0–7. Each page contains 4K words. RT–11 references each

page by the Active Page Register (APR). The APR contains the relocation constant, which controls the mapping for each page. Figure 11–8 illustrates the program virtual address space, divided into pages. Keep in mind the structure of your program in terms of how it uses the virtual address space so that you can design its overlay structure correctly and efficiently.

Figure 11–8: Program Virtual Address Space



Each overlay that is to reside in extended memory must start on one of the 4K word page boundaries. The linker automatically rounds up the size of each segment to achieve this. The linker thereby restricts you to a region reserved for the root, and a maximum of seven virtual overlay regions, each starting on a page boundary. If any of these segments extends beyond a 4K word boundary, then one fewer virtual overlay regions is available. For example, if the root is 5K words long, then the static region uses the addresses referenced by APRs 0 and 1. Only six virtual overlay regions will remain, those referenced by APRs 2 through 7.

11.4.2.2 Physical Address Space — When LINK creates the load module for a program that has overlays in extended memory, it defines how each overlay will be mapped to extended memory during run time. LINK handles extended memory overlays differently from low memory overlays. Figures 11–9 and 11–10 compare the differences.

Figure 11–9 shows the physical address space of a program that has low memory overlays. Overlay segments share each region, and each is read in from an auxiliary mass storage volume when called.

Figure 11–9: Physical Address Space for Program with Low Memory Overlays

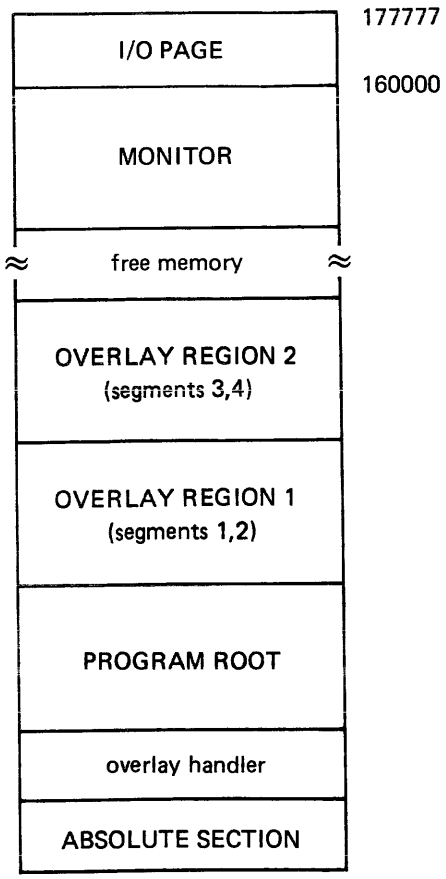
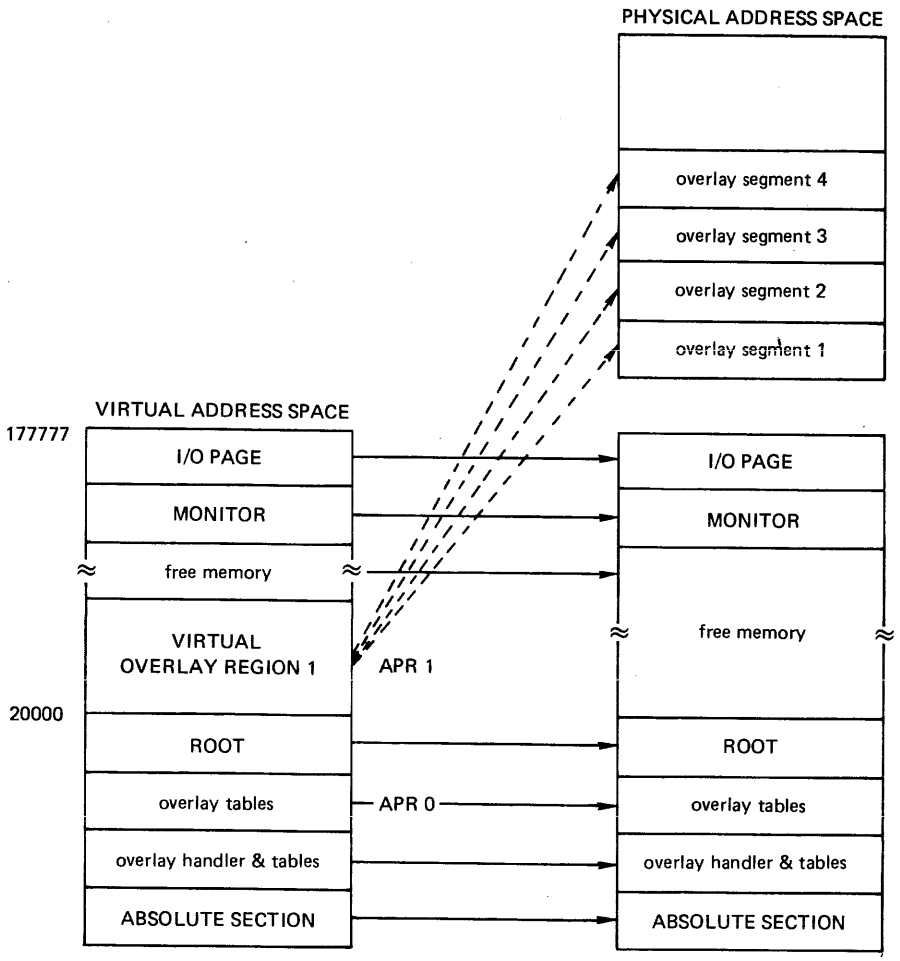


Figure 11–10: Virtual and Physical Address Space



In Figure 11–10, the diagram on the left shows the program virtual address space (0 to 177777). The diagram on the right shows the physical address space. In the program virtual address space, there is only one overlay region, and it starts on a 4K word boundary (APR 1 references this region). The regions of address space that will map to extended memory are called virtual overlay regions. Notice the arrows that point from the virtual overlay region to a number of overlay segments that appear on the right.

The overlay segments in the virtual overlay region shown use the space specified by APR 1 (20000 to 37777), but they occupy contiguous areas of extended memory, called partitions. At run time, overlay segments 1 through 4, once called, are concurrently resident in extended memory, and no further disk I/O is done to access these segments.

11.4.2.3 Virtual and Privileged Jobs — The amount of virtual address space available to your program depends on the type of program you are running. Background, foreground, and system jobs can fall into two categories: virtual and privileged.

Virtual jobs can use all 32K words of the virtual address space, but they cannot directly access the I/O page, the monitor, the vectors, or other jobs. Unless you need to access these protected areas of memory, make your jobs virtual by setting bit 10 of the JSW.

Privileged jobs also have 32K words of virtual addressing space, but by default, the protected areas (monitor, I/O page, vectors, and so on) are part of this addressing space. Just as you may lose access to protected areas if you implement your own extended memory mapping, you may lose access to the monitor and I/O page if you use extended memory overlays with a privileged job.

Virtual and privileged jobs can map to extended memory. You can use extended memory overlays with any type of virtual job (foreground, system, background) and with background privileged jobs. You cannot use extended memory overlays with privileged foreground and system jobs.

See the *RT–11 Software Support Manual* for more details on virtual and privileged jobs, and see the *RT–11 Programmer's Reference Manual* for instructions on how to make a job virtual.

11.4.2.4 Extended Memory Overlay Option (/V:n[:m]) — Use the /V option to describe your program's structure in terms of virtual overlay regions (areas of virtual address space) and partitions (areas of physical address space). The argument, *n*, represents a virtual overlay region, and *m* represents a partition. As you specify successive extended memory overlay segments in the command string, make sure that the *n* and *m* in the /V:n[:m] notation are in ascending order. The following examples show how to use the /V:n[:m] option.

In the first example, program PROG has four segments to be mapped to extended memory. The four segments are named SEG1, SEG2, SEG3, and SEG4.

```
.R LINK
*PROG=PROG//
*SEG1/V:1
*SEG2/V:1
*SEG3/V:1
*SEG4/V:1//
```

These segments map to extended memory exactly as Figure 11–10 shows. Notice how each segment fits into its own partition in extended memory. Because each segment fits into its own partition, no storage volume access is necessary to change (or swap) segments once they have been read in.

NOTE

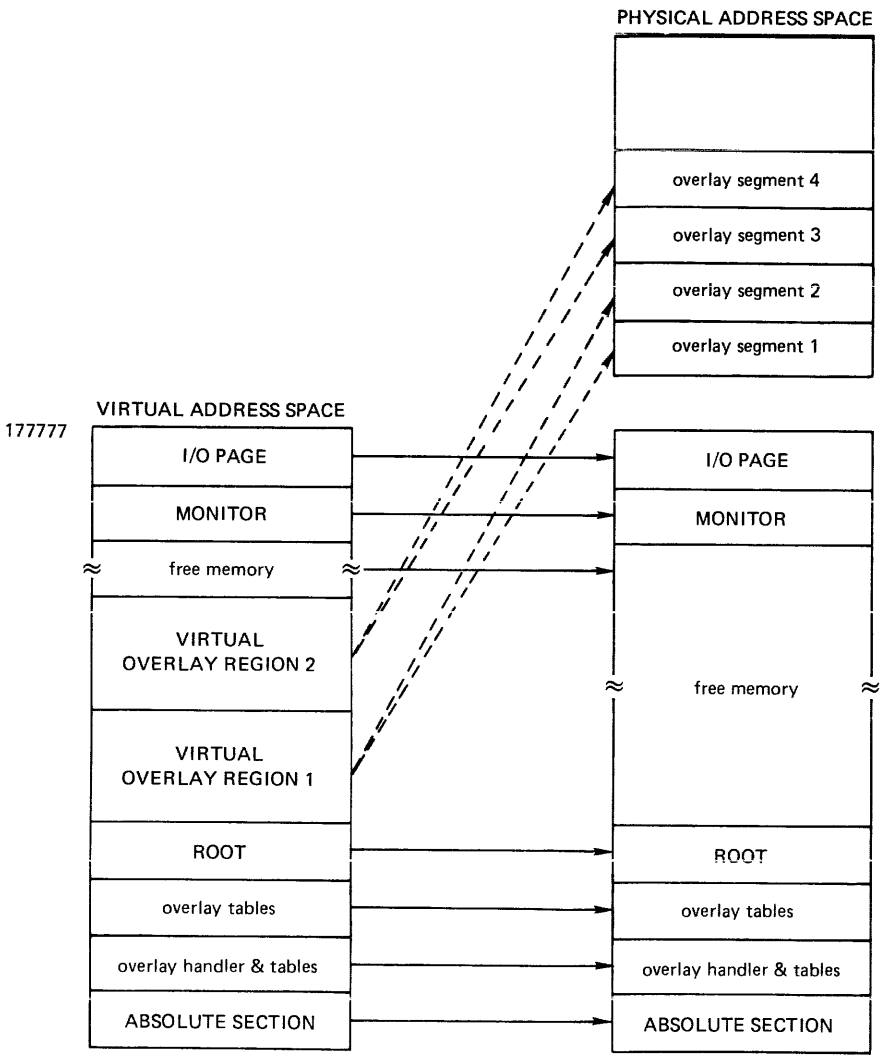
The /V:n[:m] option works differently from the /O:n option. If /O:n were used in the previous example, the four segments would share the same physical locations, obviously requiring storage volume I/O as each segment is called. With /V:n[:m], each segment from the previous example

occupies a unique area in extended memory, and no mass storage I/O is necessary after each segment is called.

The next example places the same four segments into virtual overlay regions 1 and 2. Although the program in this example uses two virtual overlay regions at run time, the segments will reside in memory the same as the segments shown in Figure 11–10. The virtual address space will be different for this example, however (see Figure 11–11). SEG1 and SEG2 use APR 1 (20000 to 37777), while SEG3 and SEG4 use APR 2 (40000 to 57777).

```
.R LINK
*PROG=PROG//
*SEG1/V:1
*SEG2/V:1
*SEG3/V:2
*SEG4/V:2//
```

Figure 11–11: Virtual and Physical Address Space

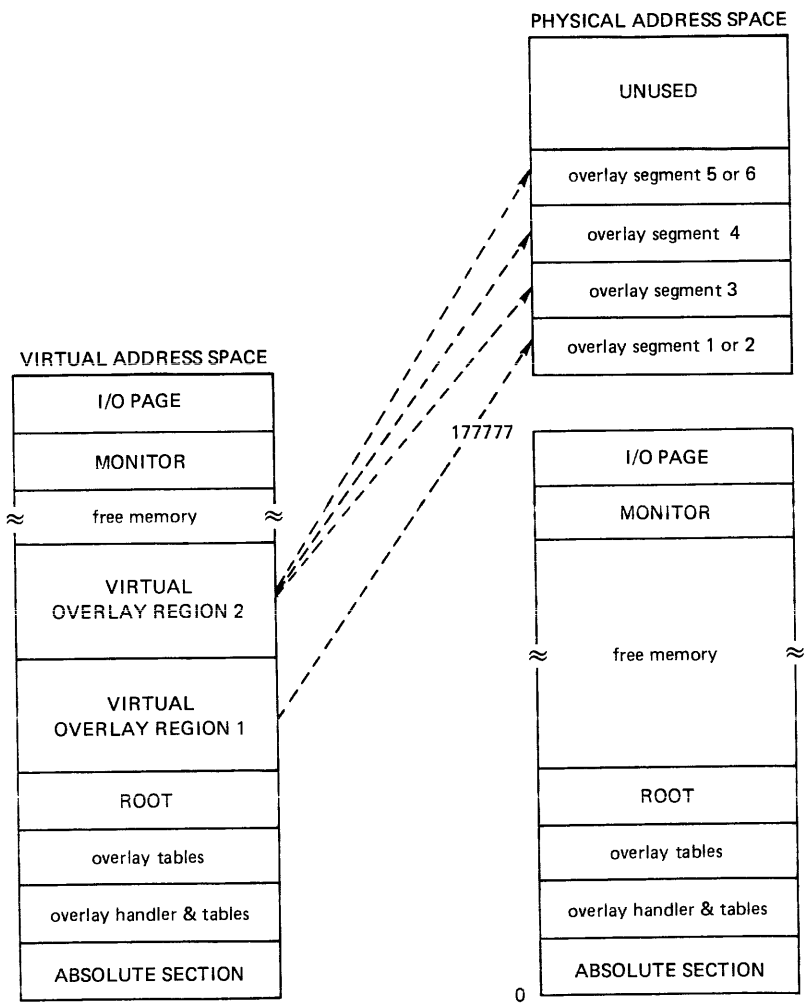


The argument, *m* in /V:n[:m], represents the partition in extended memory for the overlay segment. If you use *m*, segments can share the same partition in extended memory. That is, a segment, when called by your program, can be read in from auxiliary storage, thus overlaying the segment that currently occupies the same partition. When segments share partitions, the program requires auxiliary storage for I/O during run time, as does a program with low memory overlays.

LINK makes each partition the size of the largest segment it must accommodate. The following example generates the overlay structure shown in Figure 11–12.

```
.R LINK
*PROG=PROG//
*SEG1/V:1:1
*SEG2/V:1:1
*SEG3/V:2
*SEG4/V:2
*SEG5/V:2:1
*SEG6/V:2:1//
```

Figure 11–12: Extended Memory Partitions That Contain Sharing Segments



Notice in the specifications above that there are four segments specified for virtual overlay region 2, and that two segments share partition 1. The *m* value in /V:n:m groups segments in a region. The only reason to use the argument *m* is to create a partition that contains two or more segments. As shown in the previous example, the *m* argument is specified in ascending order within each virtual overlay region. This means you can renumber *m* from 1 for each virtual overlay region.

If you specify four segments for the same virtual overlay region, as in Example 1 below, the result is the same as if you specified Example 2. Because two segments are not specified to share the same partition, the partition order is as Example 2 shows.

Example 1

```
*SEG1/V:1
*SEG2/V:1
*SEG3/V:1
*SEG4/V:1
```

Example 2

```
*SEG1/V:1:1
*SEG2/V:1:2
*SEG3/V:1:3
*SEG4/V:1:4
```

11.4.2.5 Combining Low Memory Overlays with Extended Memory Overlays — You can combine low memory overlays and extended memory overlays in the same program structure. If you do so, however, each low memory overlay region you use makes your remaining virtual address space smaller.

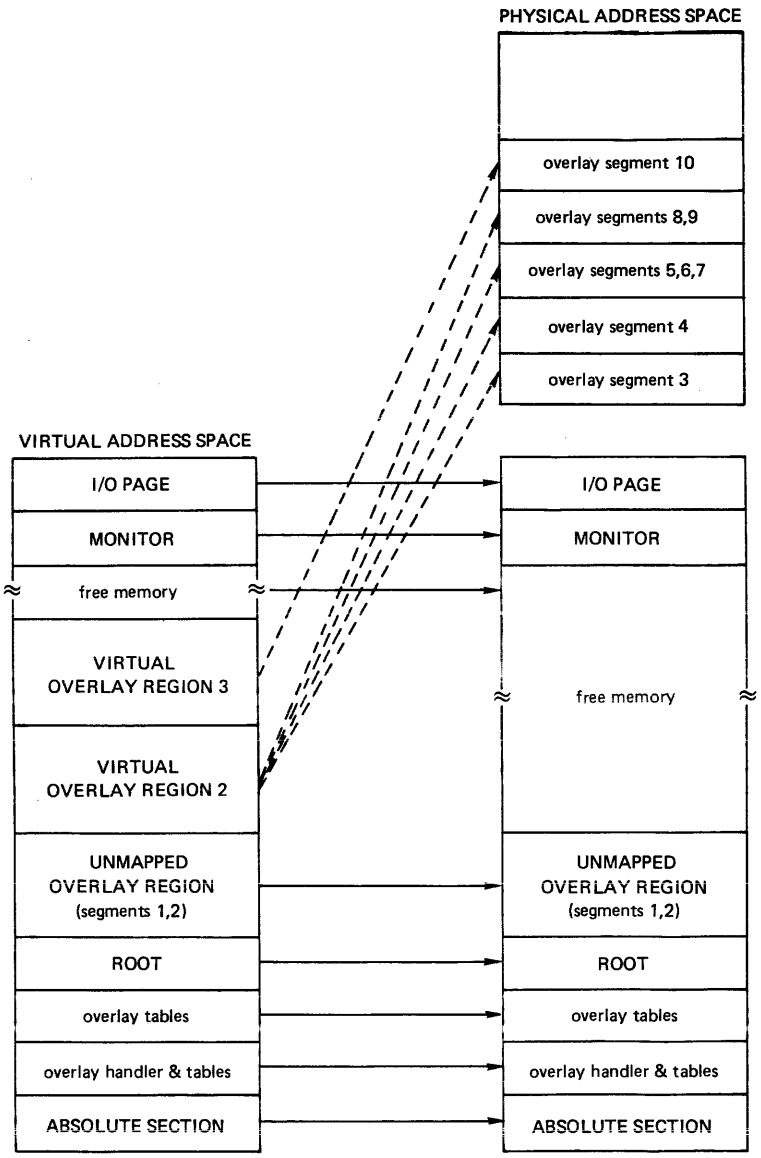
It is important to note that as you combine low memory overlays with extended memory overlays, you must list your regions in ascending order, whether or not one is a low memory overlay region and the next is a virtual region. That is, if the first overlay region is a low memory overlay region, specify it as region 1. If the next region is a virtual region, specify it as region 2. Note that you must specify low memory overlays before extended memory overlays.

The following example creates a low memory overlay region and a virtual overlay region above it.

```
.R LINK
*PROG=PROG
*SEG1/O:1
*SEG2/O:1
*SEG3/V:2
*SEG4/V:2
*SEG5/V:2:1
*SEG6/V:2:1
*SEG7/V:2:1
*SEG8/V:2:2
*SEG9/V:2:2
*SEG10/V:3
```

Figure 11–13 shows how low memory and extended memory might appear if the program from the previous example were loaded.

Figure 11–13: Memory Diagram Showing Low Memory and Extended Memory Overlays



11.4.3 Load Map

Figure 11–14 shows a sample load map for PROG.SAV, whose overlay structure is defined below.

```
*PROG,PROG=MOD0//
*MOD1/O:1
*MOD2/O:1
*MOD3/V:2
*MOD4/V:3//
```

Figure 11–14: Load Map for Program with Unmapped and Virtual Overlays

```
1  RT-11 LINK  V06.01      Load Map        Thu 08-Nov-79 14:17:26
2  PROG  .SAV      Title:  .MAIN.  Ident:  V01.00
3
4  Section  Addr   Size    Global  Value   Global  Value   Global  Value
5
6   . ABS.  000000 001000   (RW,I,GBL,ABS,OVR)
7   $OHAND  001000 000252   (RW,I,GBL,REL,CON)
8                           $OVRHV  001000  $OVRH   001004  V$READ  001034
9                           V$DONE  001046  $VDF5   001234  $VDF4   001236
10                          $VDF1   001246  $VDF2   001250
11  $OTABL  001252 000114   (RW,D,GBL,REL,OVR)
12          001366 000410   (RW,I,LCL,REL,CON)
13  MAIN    001776 000070   (RW,I,LCL,REL,CON)
14                          START   001776  RET1    002010  RET2    002014
15                          LIMIT   002024
16  LML4    002066 000026   (RW,I,GBL,REL,CON)
17                          MSGL    002066
18  LML5    002114 000026   (RW,I,GBL,REL,CON)
19                          MSGL2   002114
20  Segment size = 002142 = 561.   words
21
22 Overlay region  000001  Segment 000001
23  LML2    002144 000032   (RW,I,LCL,REL,CON)
24                          START1@ 002144
25  Segment size = 000032 = 13.    words
26
27 Overlay region  000001  Segment 000002
28  LML3    002144 000036   (RW,I,LCL,REL,CON)
29                          START2@ 002144
30  Segment size = 000036 = 15.    words
31
32 ------------------------------------------------------------------------
33
34 Virtual overlay region  000002
35 ----------------------
36
37 Partition       000001  Segment 000003
38  LML7    020002 000034   (RW,I,LCL,REL,CON)
39                          START3  020002
40  LML6    020036 000042   (RW,I,GBL,REL,CON)
41                          MSGL3   020036  RET4    020050
42  Segment size = 000076 = 31.    words
43
44 Virtual overlay region  000003
45 ----------------------
46
47 Partition       000002  Segment 000004
48  LML9    040002 000076   (RW,I,GBL,REL,CON)
49                          MSGL9 @ 040002
50  Segment size = 000076 = 31.    words
51
52
53 Transfer address = 001776, High limit = 002200 = 576.   words
54
55
56 Virtual high limit = 040076 = 8223.  words, next free address = 060000
57
58
59 Extended memory required = 000200 = 64.    words
```

Table 11–9 gives a line-by-line description of the load map above. This table makes references only to those portions of the load map that are unique to overlaid programs. For details on other parts of the load map, see Section 11.3.4.

Table 11–9: Line-by-Line Sample Load Map Description

Line	Description
7–10	$OHAND p-sect. This is the overlay handler for overlays in both low and extended memory.
11	$OTABL p-sect. This program section contains tables of data used by the overlay handler.
12	Blank p-sect. The load map for overlaid programs lists the blank p-sect, when present, after the $OHAND and $OTABL p-sects.
20	Contains data about the size of the program's root. The sections of the load map that follow provide information on the part of the program that is overlaid.
22	Header for overlay region 1, segment 1. (Low memory overlay region.)
23–24	LML2 p-sect. This is the only p-sect in segment 1. Notice in line 24 the @ character next to the global START2. This character indicates that its associated global is accessed through data contained in the overlay table p-sect, $OTABL, which is in the root.
25	Contains data on the size of segment 1.
32	Delineates the portion of the load map devoted to low memory from the portion devoted to extended memory.
34	Header for virtual overlay region 2. Note that overlay regions are numbered in ascending order, whether in low or extended memory.
37	Header for partition 1, segment 3.
41	Notice the absence of the @ character for the globals in p-sect LML6. This indicates that LML6 is not called outside segment 3.
42	Contains data on the size of overlay segment 3.
44	Header for virtual overlay region 3.
47	Header for partition 2, segment 4.
50	Contains data on the size of segment 4. Notice that segments 3 and 4 have the same length. LINK automatically rounds up the size of virtual overlay segments to multiples of 32 (decimal) words (or 100 octal bytes). LINK adds an overlay segment number word to the segment size number (the number 000076 that follows 040002 in line 48) to give the actual segment size.
53	Transfer address and high limit. The transfer address is the start address of the program. The high limit is the last low memory address used by the root and unmapped overlays.
56	Virtual high limit. Indicates the last virtual address used by the part of the program in extended memory. The next free address is the address of the next page not in use by the program.
59	Indicates the amount of extended memory required by the program. Make sure you check this figure to ensure you have adequate space for your program at runtime.

Figure 11–15 shows the extended memory overlay handler.

Figure 11–15: Extended Memory Overlay Handler

```
.SBTTL  THE RUN-TIME OVERLAY HANDLER

;+
; THE FOLLOWING CODE IS INCLUDED IN THE USER'S PROGRAM BY THE
; LINKER WHENEVER LOW MEMORY OVERLAYS ARE REQUESTED BY THE USER.
; THE RUN-TIME LOW MEMORY OVERLAY HANDLER IS CALLED BY A DUMMY
; SUBROUTINE OF THE FOLLOWING FORM:
;
;       JSR     R5,$OVRH        ;CALL TO COMMON CODE FOR LOW MEMORY OVERLAYS
;       .WORD   <OVERLAY # *6>  ;# OF DESIRED SEGMENT
;       .WORD   <ENTRY ADDR>    ;ACTUAL CORE ADDR (VIRTUAL ADDR)
;
; ONE DUMMY ROUTINE OF THE ABOVE FORM IS STORED IN THE RESIDENT PORTION
; OF THE USER'S PROGRAM FOR EACH ENTRY POINT TO A LOW MEMORY OVERLAY SEGMENT.
; ALL REFERENCES TO THE ENTRY POINT ARE MODIFIED BY THE LINKER TO BE
; REFERENCES TO THE APPROPRIATE DUMMY ROUTINE.  EACH OVERLAY SEGMENT
; IS CALLED INTO CORE AS A UNIT AND MUST BE CONTIGUOUS IN CORE.  AN
; OVERLAY SEGMENT MAY HAVE ANY NUMBER OF ENTRY POINTS, TO THE LIMITS
; OF CORE MEMORY.  ONLY ONE SEGMENT AT A TIME MAY OCCUPY AN OVERLAY REGION.
;
; IF OVERLAYS IN EXTENDED MEMORY ARE SPECIFIED, THE FOLLOWING DUMMY SUBROUTINE
; IS USED AS THE ENTRY POINT TO THE EXTENDED MEMORY OVERLAY HANDLER.
;
;       JSR     PC,$OVRHV       ;ENTRY FOR /V (EXTENDED MEMORY) OVERLAYS
;       .WORD   <OVERLAY #*6>   ;# OF DESIRED SEGMENT
;       .WORD   <VIRTUAL ENTRY ADDRESS> ;VIRTUAL ADDRESS OF SEGMENT
;
; ADDITIONAL DATA STRUCTURES IN THE EXTENDED MEMORY OVERLAY HANDLER AND THE
; OVERLAY TABLE PERMIT USE OF EXTENDED MEMORY.  ONE REGION DEFFINITION
; BLOCK IS DEFINED IN THE HANDLER, AND XM EMT'S ARE ALSO INCLUDED.  WINDOW
; DEFINITION BLOCKS FOR THE EXTENDED MEMORY PARTITIONS FOLLOW THE DUMMY
; SUBROUTINES IN THE OVERLAY TABLE.
;
; THERE IS ONE WORD PREFIXED TO EVERY OVERLAY REGION THAT IDENTIFIES THE
; SEGMENT CURRENTLY RESIDENT IN THAT OVERLAY REGION.  THIS WORD IS AN INDEX
; INTO THE OVERLAY TABLE, AND POINTS AT THE OVERLAY SEGMENT INFORMATION.
;
; UNDEFINED GLOBALS IN THE OVERLAY HANDLER MUST BE NAMED "$OVDF1" TO
; "$OVDFn" SUCH THAT A RANGE CHECK MAY BE DONE BY LINK TO DETERMINE IF
; THE UNDEFINED GLOBAL NAME IS FROM THE OVERLAY HANDLER.  A CHECK IS
; DONE ON THE .RAD50 CHARACTERS "$OV", AND THEN A RANGE CHECK IS DONE ON
; THE .RAD50 CHARATERS "DF1" TO "DFn".  THESE GLOBAL SYMBOLS DO NOT APPEAR
; ON LINK MAPS, SINCE THEIR VALUE IS NOT KNOWN UNTILL AFTER THE MAP HAS BEEN
; PRINTED.  CURRENTLY $OVDF1 TO $OVDF5 ARE IN USE.
;
; GLOBAL SYMBOLS V$READ, AND V$DONE ARE USEFULL WHEN DEBUGGING
; OVERLAID PROGRAMS.
;
;       V$READ:: WILL APPEAR IN THE LINK MAP, AND LOCATES THE .READW
;       STATEMENT IN THE OVERLAY HANDLER.
;
;       V$DONE:: WILL APPEAR IN THE LINK MAP, AND LOCATES THE FIRST
;       INSTRUCTION AFTER THE .READW IN THE OVERLAY HANDLER.
;-

.MCALL  .WDBDF,.RDBDF,.PRINT,.CRAW,.EXIT,.READW,..V1..
        ..V1..                  ;V1 FORMAT
        .WDBDF                  ;DEFINE WDB OFFSETS
        .RDBDF                  ;DEFINE RDB OFFSETS

.SBTTL  OVERLAY HANDLER CODE

.PSECT  $OHAND,GBL

.ENABL  GBL
.ENABL  LSB

;+
; THERE ARE TWO ENTRY POINTS TO THE OVERLAY HANDLER: $OVRHV FOR /V
; (EXTENDED MEMORY) OVERLAYS, AND $OVRH FOR /O (LOW MEMORY) OVERLAYS.
;-
```

(continued on next page)

```
$OVRHV::INC     (PC)+           ;SET /V OVERLAY ENTRY SWITCH
1$:     .WORD   0               ;=0 IF /O ; =1 IF /V OVERLAY ENTRY
$OVRH:: MOV     R0,-(SP)        ;/O OVERLAY ENTRY POINT
        MOV     R1,-(SP)        ;SAVE REGISTERS
        MOV     R2,-(SP)

2$:
        BR      7$              ;FIRST CALL ONLY * * *
;       MOV     @R5,R1          ;PICK UP OVERLAY NUMBER
        ADD     #$OVTAB-6,R1    ;CALC TABLE ADDR
        MOV     (R1)+,R2        ;GET FIRST ARG. OF OVERLAY SEG. ENTRY
        TST     1$              ;IS THIS /V ENTRY?
        BNE     6$              ;IF NOT EQUAL 0 THEN YES
3$:     CMP     (R5)+,@R2       ;IS OVERLAY ALREADY RESIDENT?
        BEQ     4$              ;YES, BRANCH TO IT

;+
; THE .READW ARGUMENTS ARE AS FOLLOWS:
; CHANNEL NUMBER, CORE ADDRESS, LENGTH TO READ, RELATIVE BLOCK ON DISK.
; THESE ARE USED IN REVERSE ORDER FROM THAT SPECIFIED IN THE CALL.
;-

V$READ::.READW  17,R2,@R1,(R1)+ ;READ FROM OVERLAY FILE
V$DONE::BCS     5$
4$:     MOV     (SP)+,R2        ;RESTORE USERS REGISTERS
        MOV     (SP)+,R1
        MOV     (SP)+,R0
        MOV     @R5,R5          ;GET ENTRY ADDRESS
        CLR     1$              ;CLEAR /V FLAG
        RTS     R5              ;ENTER OVERLAY ROUTINE AND RESTORE USER'S R5

5$:     EMT     376             ;SYSTEM ERROR 10 (OVERLAY I/O)
        .BYTE   0,373

6$:     MOV     R1,-(SP)        ;SAVE R1 ON STACK
        MOV     2(R1),R1        ;GET WORD LENGTH
        ASR     R1              ;AND CONVERT TO THE NUMBER
        ASR     R1              ;OF 32. WORD
        ASR     R1              ;(40 OCTAL)
        ASR     R1              ;BLOCKS FOR THE LENGTH
        ASR     R1              ;TO MAP
        MOV     R1,W.NLEN(R2)   ;SET LENGTH TO MAP

;+
; IN A REGION, RESIDENT PARTITIONS WITH THE SAME BASE ADDRESS USE DIFFERENT
; WDB'S. ONLY ONE OF THESE WINDOWS IS PRESENT AT ANY TIME. A CRAW MUST BE DONE
; TO MAKE SURE THE WINDOW BEING MAPPED IS THERE NOW. THE MAP IS DONE
; BECAUSE THE WS.MAP BIT IS SET IN THE WDB.
;-
        .CRAW   #AREA,R2        ;CREATE WINDOW AND MAP IT (WS.MAP BIT SET)
        MOV     (SP)+,R1        ;RESTORE R1
        MOV     W.NBAS(R2),R2   ;GET CORE ADDR OF OVERLAY
        BCC     3$              ;C=0 THE CRAW AND MAP ARE OK
        BR      9$              ;C=1 PRINT ERROR MESSAGE

7$:     MOV     #11501,2$       ;RESTORE SWITCH INSTR (MOV @R5,R1)
        MOV     $VDF1,R1        ;START ADDR FOR CLEAR OPERATION
8$:     CMP     R1,$VDF2        ;ARE WE DONE?
        BHIS    2$              ;HIS -> DONE, OR NO /O OVERLAYS
        CLR     (R1)+           ;CLEAR ALL LOW MEMORY OVERLAY REGIONS
        BR      6$

; ERROR MESSAGE

9$:     MOV     #MSG2,R0        ;OTHERWISE ERROR
        .PRINT                  ;AND PRINT MESSAGE
        .EXIT                   ;AND EXIT

.DSABL  LSB

.SBTTL IMPURE AREA

.ENABL  LC
.NLIST BEX
MSG2:   .ASCIZ  /VHANDL-F-Window error/
.LIST BEX
```

(continued on next page)

```
.EVEN
AREA:   .WORD   0,0             ;EMT AREA BLOCK FOR .CRAW

$VDF5:: .WORD   $OVDF5          ;POINTER TO WORD AFTER WDB'S IN OVERLAY TABLE
$VDF4:: .WORD   $OVDF4          ;POINTER TO START OF WDB'S IN OVERLAY TABLE

RGADR:  .WORD   0               ;THREE WORD REGION DEFINITION BLOCK
RGSIZ:  .WORD   $OVDF3,0        ;$OVDF3 -> SET BY LINK = SIZE OF REGION

$VDF1:: .WORD   $OVDF1          ;HIGH ADDR ROOT SEGMENT + 2 (NXT AVAIL)
$VDF2:: .WORD   $OVDF2          ;HIGH ADDR /O OVERLAYS + 2 (NXT AVAIL)

.SBTTL  $OVTAP  OVERLAY TABLE

;+
; OVERLAY TABLE STRUCTURE:
;
; LOC 64 ->     $OVTAB:
;               .WORD   <CORE ADDR>,<RELATIVE BLK>,<WORD COUNT>   /O OVERLAYS
; LOC 66 ->     .WORD   <WDB ADDR>,<RELATIVE BLK>,<WORD COUNT>    /V OVERLAYS
;               DUMMY SUBROUTINES FOR ALL OVERLAY SEGMENTS
; $VDF4 ->      WINDOW DEFINITION BLOCKS FOR EXTENDED MEMORY OVERLAYS (/V)
; $VDF5 ->      WORD AFTER THE END OF THE WINDOW DEFINITION BLOCKS (/V)
;-

.PSECT  $OTABL,D,GBL,OVR

$OVTAB:

.END
```

11.5 Option Descriptions

Full descriptions of the options summarized in Table 11–6 follow in alphabetical order.

11.5.1 Alphabetical Option (/A)

The /A option lists global symbols in program sections in alphabetical order.

11.5.2 Bottom Address Option (/B:n)

The /B:n option supplies the lowest address to be used by the relocatable code in the load module. The argument, *n*, is a six-digit unsigned octal number that defines the bottom address of the program being linked. If you do not supply a value for *n*, the linker prints

```
?LINK-F-/B No value
```

Retype the command line, supplying an even octal value.

When you do not specify /B, the linker positions the load module so that the lowest address is location 1000 (octal). If the ASECT size is greater than 1000, the size of ASECT is used.

If you supply more than one /B option during the creation of a load module, the linker uses the first /B option specification. /B is illegal when you are linking to a high address (/H). The /B option is also illegal with foreground links. These foreground modules are always linked to a bottom address of 1000 (octal).

The bottom value must be an unsigned, even, octal number. If the value is odd, the *?LINK-F-/B odd-value* error message prints. Reenter the command string specifying an unsigned, even, octal number as the argument to the /B option.

11.5.3 Continue Option (/C) or (//)

The continue option (/C) lets you type additional lines of command string input. Use the /C option at the end of the current line and repeat it on subsequent command lines as often as necessary to specify all the input modules in your program. Do not enter a /C option on the last line of input.

The following command indicates that input is to be continued on the next line; the linker prints an asterisk.

```
*OUTPUT,LP:=INPUT/C
*
```

An alternate way to enter additional lines of input is to use the // option on the first line. The linker continues to accept lines of input until it encounters another // option, which can be either on a line with input file specifications, or on a line by itself. The advantage of using the // option instead of the /C option is that you do not have to type the // option on each continuation line. This example shows how the linker itself is linked:

```
*LINK,LINK=LINKO/W//
*LINK1/O:1
*LINK2/O:1
*LINK3/O:1
*LINK4/O:1
*LINK5/O:1
*LINK6/O:1
*LINK7/O:1
*LINKEM/O:1//
```

You cannot use the /C option and the // option together in a link command sequence. That is, if you use // on the first line, you must use // to terminate input on the last line. If you use /C on the first line, use /C on all lines but the last.

11.5.4 Extend Program Section Option (/E:n)

The /E:n option allows you to extend a program section in the root to a specific value. Type the /E:n option at the end of the first command line. After you have typed all input command lines, the linker prompts with:

```
Extend section?
```

Respond with the name of the program section to be extended, followed by a carriage return. The resultant program section size is equal to or greater than the value you specify, depending on the space the object code requires. The value you specify must be an even byte value. Note that you can extend only one section.

The following example extends section CODE to 100 (octal) bytes.

```
*X,TT:=LK001/E:100
Extend section? CODE
```

11.5.5 Default FORTRAN Library Option (/F)

By indicating the /F option in the command line, you can link the FORTRAN library (FORLIB.OBJ on the system device SY:) with the other object modules you specify. You do not need to specify FORLIB explicitly. For example:

```
*FILE,LP:=AB/F
```

The object module AB.OBJ from DK: and the required routines from the FORTRAN library SY:FORLIB.OBJ are linked together to form a load module called FILE.SAV.

The linker automatically searches a default system library, SY:SYSLIB.OBJ. The library normally includes the modules that compose FORLIB. The /F option is provided only for compatibility with other versions of RT–11. You should not have to use /F.

See the *RT–11 Installation and System Generation Guide* for details on combining SYSLIB and FORLIB library files.

11.5.6 Directory Buffer Size Option (/G)

When you are using modules for your program that are from a multiple definition library, LINK has to store that library's directory in an internal buffer. Occasionally, this buffer area is too small to contain an entire directory, in which case LINK is unable to process those modules. The /G option instructs LINK to adjust the size of its directory buffer to accommodate the largest directory size of the multiple definition libraries you are using. You should use /G only when required because it slows down linking time. Use it only after an attempt to link your program failed because the buffer was too small. When a link failure of this sort occurs, LINK prints the message *?LINK-F-Library EPT too big, increase buffer with /G.*

11.5.7 Highest Address Option (/H:n)

The /H:n option allows you to specify the top (highest) address to be used by the relocatable code in the load module. The argument *n* represents an unsigned, even, octal number. If you do not specify *n*, the linker prints:

```
?LINK-F-/H no value
```

Retype the command, supplying an even octal number to be used as the value.

If you specify an odd value, the linker responds with:

```
?LINK-F-/H odd value
```

Retype the command, supplying an even octal number.

If the value is not large enough to accommodate the relocatable code, the linker prints:

```
?LINK-F-/H value too low
```

Relink the program with a larger value.

The /H option cannot be used with the /R or /Y or /B options.

NOTE

Be careful when you use the /H option. Most RT–11 programs use the free memory above the relocatable code as a dynamic working area for I/O buffers, device handlers, symbol tables, etc. The size of this area differs according to the memory configuration. Programs linked to a specific high address might not run in a system with less physical memory because there is less free memory.

11.5.8 Include Option (/I)

The /I option lets you take global symbols from any library and include them in the linking process even when they are not needed to resolve globals. This provides a method for forcing modules that are not called by other modules to be loaded from the library. All modules that you specify with /I go into the root. When you specify the /I option, the linker prints:

```
Library search?
```

Reply with the list of global symbols to be included in the load module; type a carriage return to enter each symbol in the list. A carriage return alone terminates the list of symbols.

The following example includes the global $SHORT in the load module:

```
*SCCA=RK1:SCCA/I
Library search?  $SHORT <RET>
Library search? <RET>
```

11.5.9 Memory Size Option (/K:n)

The /K:n option lets you insert a value into word 56 of block 0 of the image file. The argument *n* represents the number of 1K blocks of memory required by the program; *n* is an integer in the range 1–28. You cannot use the /K option with the /R option. The /K:n option is provided mainly for compatibility with the RSTS operating system. You should not need to use it with RT–11.

11.5.10 LDA Format Option (/L)

The /L option produces an output file in LDA format instead of memory image format. The LDA format file can be output to any device including those that are not block-replaceable, such as paper tape or cassette. It is useful for files that are to be loaded with the Absolute Loader. The default file type .LDA is assigned when you use the /L option. You cannot use the /L option with the LM overlay option (/O), the foreground link option (/R), or the XM overlay option (/V). The following example links files IN and IN2 on device DK: and outputs an LDA format file OUT.LDA to the cassette and a load map to the line printer.

```
*CT:OUT,LP:=IN,IN2/L
```

11.5.11 Modify Stack Address Option (/M[:n])

The stack address, location 42, is the address that contains the initial value for the stack pointer. The /M option lets you specify the stack address. If you use the /R:n option (foreground link) with /M, LINK ignores the value on /R:n. The argument *n* is an even, unsigned, six-digit, octal number that defines the stack address. After all input lines have been typed, the linker prints the following message if you have not specified a value for *n*:

```
Stack symbol?
```

In this case, specify the global symbol whose value is the stack address and follow with a carriage return. You must not specify a number. If you specify a nonexistent symbol, an error message prints and the stack address is set to the system default (1000 for SAV files) or to the bottom address if you used /B. If the program's absolute section extends beyond location 1000, the default stack space starts after the largest .ASECT allocation of memory.

Direct assignment (with .ASECT) of the stack address within the program takes precedence over assignment with the /M option. The statements to do this in a MACRO program are as follows:

```
.ASECT
.=42
.WORD  INITSP  ;INITIAL STACK SYMBOL VALUE
.PSECT         ;RETURN TO PREVIOUS SECTION
```

The following example modifies the stack address.

```
*OUTPUT=INPUT/M
Stack symbol? BEG <RET>
```

11.5.12 Low Memory Overlay Option (/O:n)

The /O option segments the load module so that the entire program is not memory resident at one time. This lets you execute programs that are larger than the available memory. The argument *n* is an unsigned octal number

(up to six digits in length) specifying the overlay region to which the module is assigned. The /O option must follow (on the same line) the specification of the object modules to which it applies, and only one overlay region can be specified on a command line. Overlay regions cannot be specified on the first command line; that is reserved for the root segment. You must use /C or // for continuation.

You specify co-resident overlay routines (a group of subroutines that occupy the overlay region and segment at the same time) as follows:

```
*OBJA,OBJB,OBJC/O:1/C
*OBJD,OBJE/O:2/C
 .
 .
 .
```

All modules that the linker encounters until the next /O option will be co-resident overlay routines. If you specify, at a later time, the /O option with the same value you used previously (same overlay region), then the linker opens up the corresponding overlay area for a new group of subroutines. This group occupies the same locations in memory as the first group, but it is never needed at the same time as the previous group. The following commands to the linker make R and S occupy the same memory as T (but at different times):

```
*MAIN,LP:=ROOT/C
*R,S/O:1/C
*T/O:1
```

The following example establishes two overlay regions.

```
*OUTPUT,LP:=INPUT//
*OBJA/O:1
*OBJB/O:1
*OBJC/O:2
*OBJD/O:2
*//
```

You must specify overlays in ascending order by region number. For example:

```
*A=A/C
*B/O:1/C
*C/O:1/C
*D/O:1/C
*G/O:2
```

The following overlay specification is invalid since the overlay regions are not given in ascending numerical order. An error message prints in each case, and the overlay option immediately preceding the message is ignored.

```
*X=LIBR0//
*LIBR1/O:1
*LIBR2/O:0
?LINK-W-/O or /V option error, re-enter line
*
```

In the above example, the overlay line immediately preceding the error message is ignored, and should be re-entered with an overlay region number greater than or equal to one.

11.5.13 Library List Size Option (/P:n)

The /P:n option lets you change the amount of space allocated for the library routine list. Normally, the default value allows enough space for your needs. It reserves space for approximately 170 unique library routines, which is the equivalent of specifying /P:170. (decimal) or /P:252 (octal). See the *RT–11 Installation and System Generation Guide* for details on customizing this default number for the library routine list.

The error message *?LINK-F-Library list overflow, increase size with /P* indicates that you need to allocate more space for the library routine list. You must relink the program that makes use of the library routines. Use the /P:n option and supply a value for *n* that is greater than 170.

You can use the /P:n option to correct for symbol table overflow. Specify a value for *n* that is less than 170. This reduces the space used by the library routine list and increases the space allocated for the symbol table. If the value you choose is too small, the *?LINK-F-Library list overflow, increase size with /P* message prints. In the following command, the amount of space for the library routine list is increased to 300 (decimal).

```
*SCCA=RK1:SCCA/P:300.
```

11.5.14 Absolute Base Address Option (/Q)

The /Q option lets you specify the absolute base addresses of up to eight p-sects in your program. This option is particularly handy if you are preparing your program sections in Absolute Loading format for placement in ROM storage. When you use this option in the first command line, the linker prompts you for the p-sect names and load addresses. The p-sect name must be six characters or less, and the load address must be an even octal number. Terminate each line with a carriage return. If you enter only a carriage return in response to any of the prompts, LINK ceases prompting.

If you use /E, /Y, or /U with /Q, LINK processes those options before it processes /Q.

When you use the /Q option, observe the following restrictions:

- Enter only even addresses. If you enter an odd address, no address, or invalid characters, LINK prints an error message and then prompts you again for the p-sect and load address.
- /Q is invalid with /H or /R. These options are mutually exclusive.
- LINK moves your p-sects up to the specified address; moving down might destroy code. If your address requires code to be moved down, LINK

prints an error message, ignores the p-sect for which you have specified a load address, and continues.

The following example specifies the load addresses for three p-sects.

```
*FILE,TT:=FILE,FILE1/Q/L
Load Section:Address?  PSECT1:1000 <RET>
Load Section:Address?  PSECT3:4000 <RET>
Load Section:Address?  PSECT2:2500 <RET>
Load Section:Address? <RET>
```

11.5.15 REL Format Option (/R[:n])

The /R[:n] option produces an output file in REL format for use as a foreground job with the FB or XM monitor. You cannot use .REL files under the SJ monitor. The /R option assigns the default file type .REL to the output file. The optional argument *n* represents the amount of stack space to allocate for the foreground job; it must be an even, octal number. The default value is 128 (decimal) bytes of stack space. If you also use the /M option, the value or global symbol associated with it overrides the /R value.

The following command links files FILEI.OBJ and NEXT.OBJ and stores the output on DT2: as FILEO.REL. It also prints a load map on the line printer.

```
*DT2:FILEO,LP:=FILEI,NEXT/R:200
```

You cannot use the /B, /H, or /L option with /R since a foreground REL job has a temporary bottom address of 1000 and is always relocated by FRUN. An error message prints if you attempt this. The /V and /K options are also illegal with /R.

11.5.16 Symbol Table Option (/S)

The /S option instructs the linker to allow the largest possible memory area for its symbol table at the expense of input and output buffer space. Because this makes the linking process slower, you should use the /S option only if an attempt to link a program failed because of symbol table overflow. When you use /S, do not specify a symbol table file or a map in the command string.

11.5.17 Transfer Address Option (/T[:n])

The transfer address is the address at which a program starts when you initiate execution with an R, RUN, SRUN (GET, START), or FRUN command. It prints on the last line of the load map. The /T option lets you specify the start address of the load module. The argument *n* is a six-digit unsigned octal number that defines the transfer address. If you do not specify *n* the following message prints:

```
Transfer symbol?
```

In this case, specify the global symbol whose value is the transfer address of the load module. Terminate your response with a carriage return. You cannot specify a number in answer to this message. If you specify a nonexistent symbol, an error message prints and the transfer address is set to 1 so that the program traps immediately if you attempt to execute it. If the transfer address you specify is odd, the program does not start after loading and control returns to the monitor.

Direct assignment (.ASECT) of the transfer address within the program takes precedence over assignment with the /T option. The transfer address assigned with a /T option has precedence over that assigned with an .END assembly directive. To assign the transfer address within a MACRO program, use statements similar to these:

```
        .ASECT
        .=40
        .WORD    START1  ;SYMBOL VALUE FOR TRANSFER ADDRESS
        .PSECT           ;RETURN TO PREVIOUS SECTION

START1:    .
           .
           .
```

or

```
START2:    .                ;SECONDARY STARTING ADDRESS
           .
           .
          .END     START2
```

The following example links the files LIBR0.OBJ and ODT.OBJ together and starts execution at ODT's transfer address.

```
*LBRODT,LBRODT=LIBR0,ODT/T/W//
*LIBR1/O:1
*LIBR2/O:1
*LIBR3/O:1
*LIBR4/O:1
*LIBR5/O:1
*LIBR6/O:1
*LBREM/O:1//
Transfer symbol? O.ODT
*
```

11.5.18 Round Up Option (/U:n)

The /U:n option rounds up the section you specify in the root so that the size of the root segment is a whole number multiple of the value. The argument *n* must be a power of 2. When you specify the /U:n option, the linker prompts:

```
Round section?
```

Reply with the name of the program section to be rounded, followed by a carriage return. The program section must be in the root segment. Note that

you can round only one program section. The following example rounds up section CHAR.

```
*LK007,TT:=LK007/U:200
Round section?  CHAR
```

If the program section you specify cannot be found, the linker prints *?LINK-W-Round section not found,* and the linking process continues with no rounding.

11.5.19 Extended Memory Overlay Option (/V:n[:m])

Use the /V option to create an extended memory overlay structure for your program. See Section 11.4.2 for a complete description of this option.

If you use /V on the first command line, you enable special .SETTOP features provided by the XM monitor and special .LIMIT features. When used on the first line of the command string, this option allows virtual background and foreground jobs to map a work area in extended memory with the .SETTOP programmed request. Thus, your program does not need an extended memory overlay structure to make use of the XM .SETTOP features. See the *RT–11 Programmer's Reference Manual* and the *RT–11 Software Support Manual* for more details on these features and extended memory.

11.5.20 Map Width Option (/W)

The /W option directs the linker to produce a wide load map listing. If you do not specify the /W option, the listing is wide enough for three Global Value columns (normal for paper with 80 columns). If you use the /W command, the listing is six columns wide, which is suitable for a 132-column page.

11.5.21 Bitmap Inhibit Option (/X)

The /X option instructs the linker not to output the bitmap if code is between 360 and 377 inclusive. This option is provided for compatibility with the RSTS operating system. The bitmap is stored in locations 360–377 in block 0 of the load module, and the linker normally stores the program memory usage bits in these eight words. Each bit represents one 256-word block of memory. This information is required by the R, RUN, and GET commands when loading the program; therefore, use care when you use this option.

11.5.22 Boundary Option (/Y:n)

The /Y:n option starts a specific program section in the root on a particular address boundary. Do not use this option with /H. The linker generates a whole number multiple of *n*, the value you specify, for the starting address

of the program section. The argument *n* must be a power of 2. The linker extends the size of the previous program section to accommodate the new starting address. When you have entered all the input lines, the linker prompts:

```
Boundary section?
```

Respond with the name of the program section whose starting address you are modifying. Terminate your response with a carriage return. Note that you can specify only one program section for this option. If the program section you specify cannot be found, the linker prints *?LINK-W-Boundary section not found*, and the linking process continues.

The RT–11 monitors have internal two-block overlays. The first overlay segment, OVLY0, must start on a disk block boundary:

```
*RT11SJ,SYS=BTSJ,RMSJ,KMSJ,TBSJ/Y:1000
Boundary Section?  OVLY0
```

11.5.23 Zero Option (/Z:n)

The /Z:n option fills unused locations in the load module and places a specific value in these locations. The argument *n* represents that value. This option can be useful in eliminating random results that occur when the program references uninitialized memory by mistake. The system automatically zeroes unused locations. Use the /Z:n option only when you want to store a value other than zero in unused locations. You cannot use the R, RUN, FRUN, or GET commands to load into memory a load image block of fill characters.

11.6 Linker Prompts

Some of the linker operations prompt for more information, such as the names of specific global symbols or sections. The linker issues the prompt after you have entered all the input specifications, but before the actual linking begins. Table 11–10 shows the sequence in which the prompts occur.

Table 11–10: Linker Prompting Sequence

Prompt	Option
Transfer symbol?	/T
Stack symbol?	/M
Extend section?	/E:n
Boundary section?	/Y:n
Round section?	/U:n
Load section:address?	/Q
Library search?	/I

The library search and the load section prompts can accept more than one symbol and are terminated by a carriage return in response to the prompt.

Note that if the command lines are in an indirect file and the linker encounters an end-of-file before the prompting information has been supplied, the linker prints the prompt messages on the terminal.

The following example shows how the linker prompts for information when you combine options.

```
*LK001=LK001/T/M/E:100/Y:400/U:20/I/Q
Transfer symbol? 0.ODT
Stack symbol? ST3
Extend section? CHAR
Boundary section? CODE
Round section? STKSP
Load section:address? MAIN:100000
Load section:address? <RET>
Library search? $SHORT
Library search? <RET>
*
```

Chapter 12
Librarian (LIBR)

The librarian utility program (LIBR) lets you create, update, modify, list, and maintain object library files. It also lets you create macro library files to use with the V03 and later versions of the MACRO–11 assembler.

A library file is a direct access file (a file that has a directory) that contains one or more modules of the same module type. The librarian organizes the library files so that the linker and MACRO–11 assembler can access them rapidly. Each library contains a library header, library directory (or global symbol table, or macro name table), and one or more object modules, or macro definitions. The object modules in a library file can be routines that are repeatedly used in a program, routines that are used by more than one program, or routines that are related and simply gathered together for convenience. The contents of the library file are determined by your needs. An example of a typical object library file is the default system library that the linker uses, SYSLIB.OBJ. An example of a macro library file is SYSMAC.SML, which MACRO uses to process programmed requests.

You access object modules in a library file from another program by making calls or references to their global symbols; you then link the object modules with the program that uses them, producing a single load module (see Chapter 11).

Consult the *RT–11 Software Support Manual* for more information on the internal data structure of a library file.

12.1 Calling and Using LIBR

To call the RT–11 librarian from the system device, respond to the dot (.) printed by the keyboard monitor by typing:

```
R LIBR <RET>
```

The Command String Interpreter prints an asterisk at the left margin of the console terminal when it is ready to accept a command line. Chapter 6, Command String Interpreter, describes the general syntax of the command line LIBR accepts.

Type two CTRL/C's to halt the librarian at any time (or a single CTRL/C to halt the librarian when it is waiting for console terminal input) and return control to the monitor. To restart the librarian, type R LIBR or REENTER in response to the monitor's dot.

Section 12.2 explains how to use the librarian to create and maintain object libraries; Section 12.3 describes how to create macro libraries.

Specify the LIBR command string in the following general format:

library-filespec[n],list-filespec[n] = input-filespecs/options

where:

library-filespec[n]	represents the library file to be created or updated. The optional argument *n* represents the number of blocks to allocate for the output file
list-filespec[n]	represents a listing file for the library's contents. The optional argument *n* represents the number of blocks to allocate for the listing file
input-filespec	represents the input object modules (you can specify up to six input files); it can also represent a library file to be updated
option	represents an option from Table 12–1

You specify devices and file names in the standard RT–11 command string syntax, with default file types for object libraries assigned as follows:

Object File	Default File Type
List file:	.LST
Library output file:	.OBJ
Input file (library or module):	.OBJ

If you do not specify a device, the default device (DK:) is assumed.

Each input file consists of one or more object modules and is stored on a given device under a specific file name and file type. Once you insert an object module into a library file, you no longer reference the module by the name of the file of which it was a part; instead you reference it by its individual module name. You assign this module name with the assembler with either a .TITLE statement in the assembly source program, or with the default name .MAIN. in the absence of a .TITLE statement or the subprogram name for FORTRAN routines. Thus, for example, the input file FORT.OBJ can exist on DT2: and can contain an object module called ABC. Once you insert the module into a library file, reference only ABC (not FORT.OBJ).

The input files normally do not contain main programs but rather subprograms, functions, and subroutines. The library file must never contain a FORTRAN BLOCK DATA subprogram because there is no undefined global symbol to cause the linker to load it automatically.

12.2 Option Commands and Functions for Object Libraries

You maintain object library files by using option commands. Functions that you can perform include object module deletion, insertion and replacement, library file creation, and listing of an object library file's contents.

Table 12–1 summarizes the options available for you to use with RT–11 LIBR for object libraries. The following sections, which are arranged alphabetically by option, describe the options in greater detail.

Table 12–1: LIBR Object Options

Option	Command Line	Section	Meaning
/A	first	12.2.1	Puts all globals in the directory, including all absolute global symbols.
/C	any but last	12.2.2	Command continuation; allows you to type the input specification on more than one line.
/D	first	12.2.5	Delete; deletes modules that you specify from a library file.
/E	first	12.2.6	Extract; extracts a module from a library and stores it in an OBJ file.
/G	first	12.2.6	Global deletion; deletes global symbols that you specify from the library directory.
/N	first	12.2.8	Names; includes the module names in the directory.
/P	first	12.2.9	P-sect names; includes the program section names in the directory.
/R	first	12.2.10	Replace; replaces modules in a library file. This option must follow the file specification to which it applies.
/U	first	12.2.11	Update; inserts and replaces modules in a library file. This option must follow the file specification to which it applies.
/W	first	12.2.12	Indicates wide format for the listing file.
/X	first	12.2.13	Allows multiple definitions of global entry points to appear in the library entry point table.
//	first and last	12.2.2	Command continuation; allows you to type the input specification on more than one line.

There is no option to indicate module insertion. If you do not specify an option, the librarian automatically inserts modules into the library file.

12.2.1 Include All Global and Absolute Global Symbols Option (/A)

Use the /A option when you want all the global symbols to appear in the library file's directory. When you use /A, the librarian includes in the directory all absolute global symbols, including those that have a value of 0.

Normally, the librarian includes in the directory only global entry points (labels), and not absolute global symbols.

The following example places all the global symbols from module MOD1 and MOD2 in the library directory for ALIB.OBJ.

```
*ALIB=MOD1,MOD2/A
```

12.2.2 Command Continuation Options (/C and //)

You must use a continuation option whenever there is not enough room to enter a command string on one line. The maximum number of input files that you can enter on one line is six; you can use the /C option or the // option to enter more. Type the /C option at the end of the current line and repeat it at the end of subsequent command lines as often as necessary, so long as memory is available; if you exceed memory, an error message prints. Each continuation line after the first command line can contain only input file specifications (and no other options). Do not specify a /C option on the last line of input. If you use the // option, type it at the end of the first input line and again at the end of the last input line.

The following example creates a library file on the default device (DK:) under the file name ALIB.OBJ; it also creates a listing of the library file's contents as LIBLST.LST (also on the default device). The file names of the input modules are MAIN.OBJ, TEST.OBJ, FXN.OBJ, and TRACK.OBJ, all from DT1:.

```
*ALIB,LIBLST=DT1:MAIN,TEST,FXN/C
*DT1:TRACK
```

The next example creates a library file on the default device (DK:) under the name BLIB.OBJ. It does not produce a listing. Input files are MAIN.OBJ from the default device, TEST.OBJ from RK1:, FXN.OBJ from RK0:, and TRACK.OBJ from DT1:.

```
*BLIB=MAIN//
*RK1:TEST
*RK0:FXN
*DT1:TRACK//
```

Another way of writing this command line is:

```
*BLIB=MAIN,RK1:TEST,RK0:FXN//
*DT1:TRACK
*//
```

12.2.3 Creating a Library File

To create a library file, specify a file name on the output side of a command line.

The following example creates a new library called NEWLIB.OBJ on the default device (DK:). The modules that make up this library file are in the files FIRST.OBJ and SECOND.OBJ, both on the default device.

```
*NEWLIB=FIRST,SECOND
```

12.2.4 Inserting Modules into a Library

Whenever you specify an input file without specifying an associated option, the librarian inserts the input file's modules into the library file you name on the output side of the command string. You can specify any number of input files. If you include section names (by using /P) in the global symbol table and if you attempt to insert a file that contains a global symbol or PSECT (or CSECT) having the same name as a global symbol or PSECT already existing in the library file, the librarian prints a warning message (see Section 12.2.13 for multiple definition library creation). The librarian does, however, update the library file, ignore the global symbol or section name in error, and return control to the CSI. You can then enter another command string.

Although you can insert object modules that exist under the same name (as assigned by the .TITLE statement), this practice is not recommended because of possible confusion when you need to update these modules (Sections 12.2.2.10 and 12.2.2.11 describe replacing and updating).

NOTE

> The librarian performs module insertion, replacement, deletion, merge, and update when creating the library file. Therefore, you must indicate the library file to which the operation is directed on both the input and output sides of the command line, since effectively the librarian creates a "new" output library file each time it performs one of these operations. You must specify the library file first in the input field.

The following command line inserts the modules included in the files FA.OBJ, FB.OBJ, and FC.OBJ on DT1: into a library file named DXYNEW.OBJ on the default device. The resulting library also includes the contents of library DXY.OBJ.

```
*DXYNEW=DXY,DT1:FA,FB,FC
```

The next command line inserts the modules contained in files THIRD.OBJ and FOURTH.OBJ into the library NEWLIB.OBJ.

```
*NEWLIB,LIST=NEWLIB,THIRD,FOURTH
```

Note that the resulting library (1) contains the original library plus some new modules, and (2) replaces the original library because the same name was used in this example for the input and output library.

12.2.5 Delete Option (/D)

The /D option deletes modules and all their associated global symbols from the library.

When you use the /D option, the librarian prompts:

```
Module name?
```

Respond with the name of the module to be deleted, followed by a carriage return. Continue until you have entered all modules to be deleted. Type a carriage return immediately after the *Module name?* message to terminate input and initiate execution of the command line.

The following example deletes the modules SGN and TAN from the library file TRAP.OBJ on DT3:.

```
*DT3:TRAP=DT3:TRAP/D
Module name? SGN
Module name? TAN
Module name?
```

The next example deletes the module FIRST from the library LIBFIL.OBJ; all modules in the file ABC.OBJ replace old modules of the same name in the library; it also inserts the modules in the file DEF.OBJ into the library.

```
*LIBFIL=LIBFIL/D,ABC/R,DEF
Module name? FIRST
Module name?
```

In the following example, the librarian deletes two modules of the same name from the library file LIBFIL.OBJ.

```
*LIBFIL=LIBFIL/D
Module name? X
Module name? X
Module name?
```

12.2.6 Extract Option (/E)

The /E option allows you to extract an object module from a library file and place it in an .OBJ file.

When you specify the /E option, the librarian prints:

```
Global?
```

Respond with the name of the object module you want to extract. If you specify a global name, the librarian extracts the entire module of which that global is a part. Type a carriage return to terminate the prompting for a global.

You cannot use the /E option on the same command line as another option.

The following example extracts the ATAN routine from the FORTRAN library, SYSLIB.OBJ, and stores it in a file called ATAN.OBJ on DX1:.

```
*DX1:ATAN=SYSLIB/E
Global? ATAN
Global?
```

The next example extracts the $PRINT routine from SYSLIB.OBJ and stores it on DM1: as PRINT.OBJ.

```
*DM1:PRINT=SYSLIB/E
Global? $PRINT
Global?
```

12.2.7 Delete Global Option (/G)

The /G option lets you delete a specific global symbol from a library file's directory.

When you use the /G option, the librarian prints:

```
Global?
```

Respond with the name of the global symbol you want to delete followed by a carriage return; continue until you have entered all globals to be deleted. Type a carriage return immediately after the *Global?* message to terminate input and initiate execution of the command line.

The following command instructs LIBR to delete the global symbols NAMEA and NAMEB from the directory found in the library file ROLL.OBJ on DK:.

```
*ROLL=ROLL/G
Global? NAMEA
Global? NAMEB
Global?
```

The librarian deletes globals only from the directory (and not from the library itself). Whenever you update a library file, all globals that you previously deleted are restored unless you use the /G option again to delete them. This feature lets you recover if you delete the wrong global.

12.2.8 Include Module Names Option (/N)

When you use the /N option on the first line of the command, the librarian includes module names in the directory. The linker loads modules from libraries based on undefined globals, not on module names. The linker also provides equivalent functions by using global symbols and not module names. Normally, then, it is a waste of space and a performance compromise to include module names in the directory.

If you do not include module names in the directory, the MODULE column of the directory listing is blank, unless the module requires a continuation line to print all its globals. A plus (+) sign in the MODULE column indicates continued lines. The /N option is useful mainly when you create a temporary library in order to obtain a directory listing.

If the library does not have module names in its directory, you must create a new library to include the module names. The following example illustrates how to do this. It creates a temporary new library from the current library (by specifying the null device for output) and lists its directory on the terminal. The current library OLDLIB remains unchanged.

```
*NL:TEMP,TT:=OLDLIB/N
RT-11 LIBRARIAN V03.10  TUE 03-MAY-79 20:36:41
NL:TEMP.OBJ             TUE 03-MAY-79 20:36:40

MODULE          GLOBALS         GLOBALS         GLOBALS

IRAD50          IRAD50          RAD50
JMUL            JMUL
LEN             LEN
SUBSTR          SUBSTR
JADD            JADD
JCMP            JCMP
```

12.2.9 Include P-section Names Option (/P)

The librarian does not include program section names in the directory unless you use the /P option on the first line of the command. The linker does not use section names to load routines from libraries — in fact, including the names can decrease linker performance. Including program section names also causes a conflict in the library directory and subsequent searches, since the librarian treats section names and global symbols identically.

This option is provided for compatibility with RT–11 V2C. DIGITAL recommends that you avoid using it with later versions of RT–11.

12.2.10 Replace Option (/R)

Use the /R option to replace modules in a library file. The /R option replaces existing modules in the library file you specify as output with the modules of the same names contained in the file(s) you specify as input. In the command string, enter the input library file before the files used in the replacement operation.

If an old module does not exist under the same name as an input module, or if you specify the /R option on a library file, the librarian prints an error message followed by the module name and ignores the replace command. The /R option must follow each input file name containing modules for replacement.

The following command line indicates that the modules in the file INB.OBJ are to replace existing modules of the same names in the library file TFIL.OBJ. The object modules in the files INA.OBJ and INC.OBJ are to be added to TFIL. All files are stored on the default device DK:.

```
*TFIL=TFIL,INA,INB/R,INC
```

The same operation occurs in the next command as in the preceding example, except that this updated library file is assigned the new name XFIL.

```
*XFIL=TFIL,INA,INB/R,INC
```

12.2.11 Update Option (/U)

The /U option lets you update a library file by combining the insert and replace functions. If the object modules that compose an input file in the command line already exist in the library file, the librarian replaces the old modules in the library file with the new modules in the input file. If the object modules do not already exist in the library file, the librarian inserts those modules into the library. (Note that some of the error messages that might occur with separate insert and replace functions do not print when you use the update function.) /U must follow each input file that contains modules to be updated. Specify the input library file before the input files in the command line.

The following command line instructs the librarian to update the library file BALIB.OBJ on the default device. First the modules in FOLT.OBJ and BART.OBJ replace old modules of the same names in the library file, or if none already exist under the same names, the modules are inserted. The modules from the file TAL.OBJ are then inserted; an error message prints if the name of the module in TAL.OBJ already exists.

```
*BALIB=BALIB,FOLT/U,TAL,BART/U
```

In the next example, there are two object modules of the same name, X, in both Z and XLIB; these are first deleted from XLIB so that both the modules called X in file Z are correctly placed in the library. Globals SEC1 and SEC2 are also deleted from the directory but automatically return the next time the library XLIB.OBJ is updated.

```
*XLIB=XLIB/D,Z/U/G
Module name?  X
Module name?  X
Module name?
Global?  SEC1
Global?  SEC2
Global?
```

12.2.12 Wide Option (/W)

The /W option gives you a wider listing if you request a listing file. The wider listing has six Global columns instead of three, as in the normal list-

ing. This is useful if you list the directory on a line printer or a terminal that has 132 columns.

12.2.13 Creating Multiple Definition Libraries Option (/X)

The /X option lets you create libraries that can have more than one definition for a global entry point. These libraries are called multiple definition libraries. They are processed differently from libraries that contain only one definition for each global entry point name that appears in the library's directory (for more information on processing multiple definition libraries, see Chapter 11).

In multiple definition libraries, two library modules may use the same global entry point name, and both definitions may appear in the entry point table (EPT). At least one entry point name should be unique in each module so that you can easily identify it.

When you use the /X option, the librarian does not issue the *?LIBR-W-Illegal insert of AAAAAA* message when it encounters a duplicate global symbol name, and the global name will appear in the directory for each module that defines it. In addition, the /X option causes the librarian to turn on the /N option (see Section 12.2.8).

The following example creates the multiple definition library MLTLIB from modules MOD1, MOD2, and MOD3 and lists the library on the terminal. Since MOD3 contains only absolute global symbols, this example must also use the /A option.

```
*MLTLIB,TT:=MOD1,MOD2,MOD3/X/A

RT-11 LIBRARIAN V03.10  THU 25-APR-79 09:45:31
DK:MLTLIB.OBJ           THU 25-APR-79 09:45:31

MODULE          GLOBALS         GLOBALS         GLOBALS

MOD1            OMA$R           SWP$            ATP$
MOD2            ATP$            OMA$R           MER$CR
                LBM
MOD3            ATP$            OMA$R           MER$CR
                ENTZ
```

12.2.14 Listing the Directory of a Library File

You can request a listing of the contents of a library file (the global symbol table) by indicating both the library file and a list file in the command line. Since a library file is not being created or updated, you do not need to indicate the file name on the output side of the command line; however, you must use a comma to designate a null output library file.

The command syntax is as follows:

*,LP:=library-filespec

or

*,list-filespec = library-filespec

where:

library-filespec	represents the existing library file
LP:	indicates that the listing is to be sent directly to the line printer (or terminal, if you use TT:)
list-filespec	represents a list file of the library file's contents

The following command outputs to DT2: as LIST.LST a listing of all modules in the library file LIBFIL.OBJ, which is stored on the default device.

```
*,DT2:LIST=LIBFIL
```

The next command sends to the line printer a listing of all modules in the library file FLIB.OBJ, which is stored on the default device.

```
*,LP:=FLIB
```

Here is a sample section of a large directory listing:

```
*,TT:=SYSLIB
RT-11 LIBRARIAN V03.10   TUE 03-MAY-79 21:01:01
DK:SYSLIB.OBJ            TUE 03-MAY-79 20:59:47

MODULE           GLOBALS          GLOBALS          GLOBALS

                 DCO$             ECO$             FCO$
+                GCO$             RCI$
                 DIC$IS           DIC$MS           DIC$PS
+                DIC$SS           $DIVC            $DVC
                 ADD$IS           ADD$MS           ADD$PS
+                ADD$SS           SUD$IS           SUD$MS
+                SUD$PS           SUD$SS           $ADD
```

The first line of the listing file shows the version of the librarian that was used and the current date and time. The second line prints the library file name and the date and time the library was created. Each line in the rest of the listing shows only the globals that appear in a particular module. If a module contains more global symbol names than can print on one line, a new line will be started with a plus (+) sign in column 1 to indicate continuation.

If you request a listing of a library file that was created with the /X or /N option, the listing includes module names under the MODULE heading.

12.2.15 Merging Library Files

You can merge two or more library files under one file name by specifying in a single command line all the library files to be merged. The librarian does not delete the individual library files following the merge unless the output file name is identical to one of the input file names.

The command syntax is as follows:

*library-filespec = input-filespecs

where:

library-filespec	represents the library file that will contain all the merged files (If a library file already exists under this name, you must also indicate it in the input side of the command line so that it is included in the merge.)
input-filespec	represents a library file to be merged

Thus, the following command combines library files MAIN.OBJ, TRIG.OBJ, STP.OBJ, and BAC.OBJ under the existing library file name MAIN.OBJ; all files are on the default device DK:. Note that this replaces the old contents of MAIN.OBJ.

```
*MAIN=MAIN,TRIG,STP,BAC
```

The next command creates a library file named FORT.OBJ and merges existing library files A.OBJ, B.OBJ, and C.OBJ under the file name FORT.OBJ.

```
*FORT=A,B,C
```

NOTE

Library files that you combine using PIP are illegal as input to both the librarian and the linker.

12.2.16 Combining Library Option Functions

You can request two or more library functions in the same command line, with the exception of the /E and /M options, which cannot be specified on the same command line with any other option. The librarian performs functions (and issues appropriate prompts) in the following order:

1. /C or //
2. /D
3. /G
4. /U
5. /R
6. Insertions
7. Listing

Here is an example that combines options:

```
*FILE,LP:=FILE/D,MODX,MODY/R
Module name?  XYZ
Module name?  A
Module name?
```

The librarian performs the functions in this example in the following order:

1. Deletes modules XYZ and A from the library file FILE.OBJ.
2. Replaces any duplicate of the modules in the file MODY.OBJ.
3. Inserts the modules in the file MODX.OBJ.
4. Lists the directory of FILE.OBJ on the line printer.

12.3 Option Commands and Functions for Macro Libraries

The librarian lets you create macro libraries. A macro library works with the V03 and later MACRO–11 assembler to reduce macro search time.

The .MACRO directive produces the entries in the library directory (macro names). LIBR does not maintain a directory listing file for macro libraries; you can print the ASCII input file to list the macros in the library.

The default input and output file type for macro files is .MAC.

If you give the library file the same name as one of the input files, the librarian prints the error message: *?LIBR-F-Output and input filnames the same.*

The librarian removes all comments from your source input file except for those within a macro (that is, between a .MACRO and .ENDM pair of directives). Because comments take up space during the assembly and in the library, remove them from the macros wherever possible before creating a macro library, if saving space and shortening assembly time are important to you.

Table 12--2 summarizes the options you can use with macro libraries.

The options are explained in detail in the following two sections.

Table 12–2: LIBR Macro Options

Option	Command Line	Section	Meaning
/C	any but last	12.3.1	Command continuation; allows you to type the input specification on more than one line.
/M[:n]	first	12.3.2	Macro; creates a macro library from the ASCII input file containing .MACRO directives.
//	first and last	12.3.1	Command continuation; allows you to type the input specification on more than one line.

12.3.1 Command Continuation Options (/C or //)

These options are the same for macro libraries as for object libraries. See Section 12.2.2.

12.3.2 Macro Option (/M[:n])

The /M[:n] option creates a macro library file from an ASCII input file that contains .MACRO directives. The optional argument *n* determines the amount of space to allocate for the macro name directory by representing the number of macros you want the directory to hold. Remember that *n* is interpreted as an octal number; you must follow *n* by a decimal point (*n.*) to indicate a decimal number. Each 64 macros occupy one block of library directory space. The default value for *n* is 128, enough space for 128 macros, which will use 2 blocks for the macro name table.

The command syntax is as follows:

*library-filespec = input-filespec/M[:n]

where:

library-filespec	represents the macro library to be created
input-filespec	represents the ASCII input file that contains .MACRO definitions
/M[:n]	is the macro option

The continuation options (/C or //) are the only options you can use with the macro option.

The following example creates the macro library SYSMAC.SML from the ASCII input file SYSMAC.MAC. Both files are on device DK:.

```
*SYSMAC.SML=SYSMAC/M
```

Chapter 13
DUMP

The DUMP program prints on the console or line printer, or writes to a file, all or any part of a file as words or bytes (in octal), ASCII characters, and/or Radix–50 characters. DUMP is particularly useful for examining directories and files that contain binary data.

13.1 Calling and Using DUMP

To call the DUMP program from the system device, respond to the dot (.) printed by the keyboard monitor by typing:

```
R DUMP <RET>
```

The Command String Interpreter prints an asterisk at the left margin of the console terminal when it is ready to accept a command line. If you respond to the asterisk by typing only a carriage return, DUMP prints its current version number.

You can type CTRL/C to halt DUMP and return control to the monitor when DUMP is waiting for input from the console terminal. You must type two CTRL/C's to abort DUMP at any other time. To restart DUMP, type R DUMP or REENTER in response to the monitor's dot. Chapter 6, Command String Interpreter, describes the general syntax of the command line that DUMP accepts. If you do not specify an output file, the listing prints on the line printer. If you do not specify a file type for an output file, the system uses .DMP.

13.2 Options

Table 13–1 summarizes the options that are valid for DUMP.

Table 13–1: DUMP Options

Option	Explanation
/B	Outputs bytes, in octal.
/E:n	Ends output at block number *n*, where *n* is an octal block number.
/G	Ignores input errors.
/N	Suppresses ASCII output.
/O:n	Outputs only block number *n*, where *n* is an octal block number. With this option, you can dump only one block for each command line.

(continued on next page)

Table 13–1: DUMP Options(Cont.)

Options	Explanation
/S:n	Starts output with block number *n*, where *n* is an octal block number. For random-access devices, *n* may not be greater than the number of blocks in the file.
/T	Defines a tape as non-file-structured.
/W	Outputs words, in octal (the default mode).
/X	Outputs Radix–50 characters.

ASCII characters are always dumped unless you type /N.

If you specify an input file name, the block numbers (*n*) you supply are relative to the beginning of that file. If you do not specify a file name, that is, if you are dumping a device, the block numbers are the absolute (physical) block numbers on that device. Remember that the first block of any file or device is block 0.

NOTE

DUMP does not print data from track 0 of diskettes.

DUMP handles operations that involve magtape and cassette differently from operations involving random-access devices.

If you dump an RT–11 file-structured tape and specify only a device name in the input specification, DUMP reads only as far as the logical EOF1. Logical end-of-tape is indicated by an end-of-file label followed by two tape marks. For non-file-structured tape, logical end-of-tape is indicated by two consecutive tape marks. If you dump a cassette and specify only the device name in the input specification, the results are unpredictable. For magtape dumps, tape mark messages appear in the output listing as DUMP encounters them on the tape.

If you use /S:n with magtape, *n* can be any positive value. However, an error can occur if *n* is greater than the number of blocks written on the tape. For example, if a tape has 100 written blocks and *n* is 110, an error can occur if DUMP accesses past the 100th block. If you specify /E:n, DUMP reads the tape from its starting position (block 0, unless you specify otherwise) to block number *n* or to logical end-of-tape, whichever comes first.

13.3 Example Commands and Listings

This section includes sample DUMP commands and the listings they produce.

The following command string directs DUMP to print, in words, information contained in block 1 of the file DMPX.SAV stored on device DK:.

```
*DMPX.SAV/O:1
```

```
DMPX.SAV/O:1
BLOCK NUMBER  00001
000/ 000000 042062 000000 000000 000000 000000 000000 000000 *..2D............*
020/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
040/ 000000 000000 001002 000000 000000 003356 001010 001104 *..........n...D.*
060/ 000014 000400 004001 000000 000000 000000 000000 000000 *................*
100/ 000000 000000 045504 043072 046111 030505 044456 046523 *....DK:FILE1.ISM*
120/ 000000 046061 000000 000000 000000 000000 000000 000000 *..1L............*
140/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
160/ 000000 000000 000000 000000 000000 001356 002000 001234 *..........n.....*
200/ 000000 000000 000000 000000 000000 001000 000000 000000 *................*
220/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
240/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
260/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
300/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
320/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
340/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
360/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
400/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
420/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
440/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
460/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
500/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
520/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
540/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
560/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
600/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
620/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
640/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
660/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
700/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
720/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
740/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
760/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
```

In the printout above, the heading shows which block of the file follows. The numbers in the leftmost column indicate the byte offset from the beginning of the block. Remember that these are all octal values and that there are two bytes per word. The words that were dumped appear in the next eight columns. The rightmost column contains the ASCII equivalent of each word. DUMP substitutes a dot (.) for nonprinting codes, such as those for control characters.

The next command dumps block 1 of file PIP.SAV. The /N option suppresses ASCII output.

```
*PIP.SAV/N/O:1

SY:PIP.SAV/N/O:1
BLOCK NUMBER  00001
000/ 060502 010046 010146 010246 000422 062701 001100 012102
020/ 022512 001406 012100 005046 011146 010246 104217 103405
040/ 012602 012601 012600 011505 000205 104376 175400 012767
060/ 011501 177724 016701 000012 005021 020167 000006 103774
100/ 000743 005562 015260 005562 000006 002112 005562 000013
120/ 003147 014100 000022 000211 014100 000023 000423 014100
140/ 000025 000470 004537 001002 000006 006456 004537 001002
160/ 000014 005764 004537 001002 000022 014102 004537 001002
200/ 000030 014102 004537 001002 000036 014120 000000 000000
220/ 000000 000000 000000 000000 000000 000000 000000 000000
240/ 000000 000000 000000 000000 000000 000000 000000 000000
260/ 000000 000000 000000 000000 000000 000000 000000 000000
300/ 000000 000000 000000 001000 015260 000000 000000 000000
320/ 000000 000000 000000 000000 000000 000000 000000 000000
340/ 000000 000000 000000 000000 000000 000000 000000 000000
```

(continued on next page)

```
360/ 000000 000000 000000 000000 000000 000000 000000 000000
400/ 000000 000000 000000 000000 000000 000000 000000 000000
420/ 000000 000000 002056 002063 003452 000000 005020 000000
440/ 000000 000000 000000 000000 000000 000000 000000 000000
460/ 000000 000000 000000 000000 000000 000000 000000 000000
500/ 000000 000000 000000 000000 000000 000000 000000 000000
520/ 000000 000000 000000 000000 000000 000000 000000 000000
540/ 000000 000000 000000 000000 000000 000000 000000 000000
560/ 000000 000000 000000 000000 000000 000000 000000 000000
600/ 000000 000000 000000 000000 000000 000000 000000 000000
620/ 000000 000000 000000 000000 000000 000000 000000 000000
640/ 000000 000000 000000 000000 000000 000000 000000 000000
660/ 000000 000000 000000 000000 000000 000000 000000 000000
700/ 000000 000000 000000 000000 000000 000000 000000 000000
720/ 000000 000000 000000 000000 000000 000000 000000 000000
740/ 000000 000000 000000 000000 000000 000000 000000 000000
760/ 000000 000000 000000 000000 000000 000000 000000 000000
```

The following command dumps block 1 of SYSMAC.MAC in bytes. ASCII equivalents appear underneath each byte.

```
*SYSMAC.MAC/B/O:1

SY:SYSMAC.MAC/B/O:1
BLOCK NUMBER  00001
000/ 120 040 117 106 040 040 124 110 105 040 040 123 117 106 124 127
     P       O   F           T   H   E           S   O   F   T   W
020/ 101 122 105 040 040 111 123 040 040 110 105 122 105 102 131 015
     A   R   E           I   S           H   E   R   E   B   Y   .
040/ 012 073 040 124 122 101 116 123 106 105 122 122 105 104 056 015
     .   ;       T   R   A   N   S   F   E   R   R   E   D   .   .
060/ 012 073 015 012 073 040 124 110 105 040 111 116 106 117 122 115
     .   ;   .   .   ;       T   H   E       I   N   F   O   R   M
100/ 101 124 111 117 116 040 111 116 040 124 110 111 123 040 123 117
     A   T   I   O   N       I   N       T   H   I   S       S   O
120/ 106 124 127 101 122 105 040 111 123 040 123 125 102 112 105 103
     F   T   W   A   R   E       I   S       S   U   B   J   E   C
140/ 124 040 124 117 040 103 110 101 116 107 105 040 040 127 111 124
     T       T   O       C   H   A   N   G   E           W   I   T
160/ 110 117 125 124 040 040 116 117 124 111 103 105 015 012 073 040
     H   O   U   T           N   O   T   I   C   E   .   .   ;
200/ 101 116 104 040 040 123 110 117 125 114 104 040 040 116 117 124
     A   N   D           S   H   O   U   L   D           N   O   T
220/ 040 040 102 105 040 040 103 117 116 123 124 122 125 105 104 040
         B   E           C   O   N   S   T   R   U   E   D
240/ 040 101 123 040 040 101 040 103 117 115 115 111 124 115 105 116
         A   S           A       C   O   M   M   I   T   M   E   N
260/ 124 040 102 131 040 104 111 107 111 124 101 114 040 105 121 125
     T       B   Y       D   I   G   I   T   A   L       E   Q   U
300/ 111 120 115 105 116 124 015 012 073 040 103 117 122 120 117 122
     I   P   M   E   N   T   .   .   ;       C   O   R   P   O   R
320/ 101 124 111 117 116 056 015 012 073 015 012 073 040 104 111 107
     A   T   I   O   N   .   .   .   ;   .   .   ;       D   I   G
340/ 111 124 101 114 040 101 123 123 125 115 105 123 040 116 117 040
     I   T   A   L       A   S   S   U   M   E   S       N   O
360/ 122 105 123 120 117 116 123 111 102 111 114 111 124 131 040 106
     R   E   S   P   O   N   S   I   B   I   L   I   T   Y       F
400/ 117 122 040 124 110 105 040 125 123 105 040 117 122 040 040 122
     O   R       T   H   E       U   S   E       O   R           R
420/ 105 114 111 101 102 111 114 111 124 131 040 040 117 106 040 040
     E   L   I   A   B   I   L   I   T   Y           O   F
440/ 111 124 123 015 012 073 040 123 117 106 124 127 101 122 105 040
     I   T   S   .   .   ;       S   O   F   T   W   A   R   E
460/ 117 116 040 105 121 125 111 120 115 105 116 124 040 127 110 111
     O   N       E   Q   U   I   P   M   E   N   T       W   H   I
```

(continued on next page)

```
500/ 103 110 040 111 123 040 116 117 124 040 123 125 120 120 114 111
     C   H       I   S       N   O   T       S   U   P   P   L   I
520/ 105 104 040 102 131 040 104 111 107 111 124 101 114 056 015 012
     E   D       B   Y       D   I   G   I   T   A   L   .   .   .
540/ 014 056 115 101 103 122 117 040 056 056 126 061 056 056 015 012
     .   .   M   A   C   R   O       .   .   V   1   .   .   .   .
560/ 056 115 103 101 114 114 011 056 056 056 103 115 060 054 056 056
     .   M   C   A   L   L   .   .   .   .   C   M   0   ,   .   .
600/ 056 103 115 061 054 056 056 056 103 115 062 054 056 056 056 103
     .   C   M   1   ,   .   .   .   C   M   2   ,   .   .   .   C
620/ 115 063 054 056 056 056 103 115 064 054 056 056 056 103 115 065
     M   3   ,   .   .   .   C   M   4   ,   .   .   .   C   M   5
640/ 054 056 056 056 103 115 066 015 012 056 056 056 126 061 075 061
     ,   .   .   .   C   M   6   .   .   .   .   .   V   1   =   1
660/ 015 012 056 105 116 104 115 015 012 015 012 056 115 101 103 122
     .   .   .   E   N   D   M   .   .   .   .   .   M   A   C   R
700/ 117 040 056 056 126 062 056 056 015 012 056 115 103 101 114 114
     O       .   .   V   2   .   .   .   .   .   M   C   A   L   L
720/ 011 056 056 056 103 115 060 054 056 056 056 103 115 061 054 056
     .   .   .   .   C   M   0   ,   .   .   .   C   M   1   ,   .
740/ 056 056 103 115 062 054 056 056 056 103 115 063 054 056 056 056
     .   .   C   M   2   ,   .   .   .   C   M   3   ,   .   .   .
760/ 103 115 064 054 056 056 056 103 115 065 054 056 056 056 103 115
     C   M   4   ,   .   .   .   C   M   5   ,   .   .   .   C   M
```

The last example shows block 6 (the directory) of device RK0:. The output is in octal words with Radix–50 equivalents below each word.

```
*RK0:/N/X/0:6
```

```
RK0:/N/X/0:6
BLOCK NUMBER  00006
000/ 000020 000002 000004 000000 000046 002000 075131 062000
       P      B      D               8      YX     SWA    P
020/ 075273 000031 000000 027147 002000 071677 142302 075273
     SYS      Y             GP9     YX     RT1    1SJ    SYS
040/ 000103 000000 027147 002000 071677 141262 075273 000120
      AS            GP9     YX     RT1    1FB    SYS     B
060/ 000000 027147 002000 071677 141034 075273 000100 000000
            GP9     YX     RT1    1BL    SYS     AX
100/ 027147 002000 100040 000000 075273 000002 000000 027147
     GP9     YX     TT             SYS      B             GP9
120/ 002000 016040 000000 075273 000003 000000 027147 002000
      YX     DT             SYS      C             GP9     YX
140/ 015600 000000 075273 000003 000000 027147 002000 016300
     DP             SYS      C             GP9     YX     DX
160/ 000000 075273 000003 000000 027147 002000 016350 000000
            SYS      C             GP9     YX     DY
200/ 075273 000004 000000 027147 002000 070560 000000 075273
     SYS      D             GP9     YX     RF             SYS
220/ 000003 000000 027147 002000 071070 000000 075273 000003
       C            GP9     YX     RK             SYS      C
240/ 000000 027147 002000 015340 000000 075273 000004 000000
            GP9     YX     DL             SYS      D
260/ 027147 002000 015410 000000 075273 000005 000000 027147
     GP9     YX     DM             SYS      E             GP9
300/ 002000 015770 000000 075273 000003 000000 027147 002000
      YX     DS             SYS      C             GP9     YX
320/ 014640 000000 075273 000005 000000 027147 002000 046600
     DD             SYS      E             GP9     YX     LP
340/ 000000 075273 000002 000000 027147 002000 046770 000000
            SYS      B             GP9     YX     LS
```

(continued on next page)

```
360/ 075273 000002 000000 027147 002000 012620 000000 075273
     SYS      B           GP9      YX     CR           SYS
400/ 000003 000000 027147 002000 052070 000000 075273 000011
       C           GP9      YX     MS           SYS      I
420/ 000000 027547 002000 052150 014400 075273 000003 000000
            GWO      YX     MTH     D      SYS      C
440/ 027147 002000 015173 052177 012445 000011 000000 027547
     GP9      YX     DIS     MT1     COM      I           GWO
460/ 002000 051520 014400 075273 000004 000000 027147 002000
       YX     MMH     D      SYS      D           GP9      YX
500/ 015173 052200 012445 000010 000000 027547 002000 052100
     DIS     MT2     COM      H           GWO      YX     MSH
520/ 014400 075273 000004 000000 027147 002000 054540 000000
       D     SYS      D           GP9      YX     NL
540/ 075273 000002 000000 027147 002000 062170 000000 075273
     SYS      B           GP9      YX     PC           SYS
560/ 000002 000000 027147 002000 062240 000000 075273 000003
       B           GP9      YX     PD           SYS      C
600/ 000000 027147 002000 012740 000000 075273 000005 000000
            GP9      YX     CT           SYS      E
620/ 027147 002000 006250 000000 075273 000007 000000 027147
     GP9      YX     BA           SYS      G           GP9
640/ 002000 016130 000000 073376 000051 000000 027147 002000
       YX     DUP         SAV      AA           GP9      YX
660/ 023752 050574 073376 000023 000000 027147 002000 070533
     FOR     MAT     SAV      S           GP9      YX     RES
700/ 060223 073376 000017 000000 027147 002000 015172 000000
     ORC     SAV      O           GP9      YX     DIR
720/ 073376 000021 000000 027147 002000 075273 050553 074324
     SAV      Q           GP9      YX     SYS     MAC     SML
740/ 000052 000000 027147 002000 017751 076400 073376 000023
       AB          GP9      YX     EDI     T       SAV      S
760/ 000000 027147 002000 042614 000000 073376 000073 000000
            GP9      YX     KED          SAV      AS
```

Chapter 14
FILEX

The file exchange program (FILEX) is a general file transfer program that converts files from one format to another so that you can use them with various operating systems. You can initiate transfers between any block-replaceable RT–11 directory-structured device and any device listed in Table 14–1.

Table 14–1: Supported FILEX Devices

Device	Valid as Input	Valid as Output
PDP–11 DOS/BATCH DECtape	X	X
DOS/BATCH disk	X	
RSTS DECtape	X	X
DECsystem–10 DECtape	X	
Interchange Diskette (RX01, RX02 single-density)	X	X

FILEX does not support magtapes, cassettes, or double-density diskettes in any operation. Note that you can transfer only one file at a time to interchange diskette format.

Section 4.2 of this manual describes how to use wildcards, which you can use in the FILEX command string. You can use wildcards when transferring from interchange to RT–11 format. However, you cannot use embedded wildcards in any file name or file type. For example, the following line represents a valid file specification.

```
**.MAC
```

The next line is an illegal file specification for FILEX.

```
*T%ST.MAC
```

14.1 File Formats

FILEX can transfer files created by four different operating systems: RT–11, DECsystem–10, universal interchange format (IBM) system (see the *RT–11*

Software Support Manual), and DOS/BATCH (PDP–11 Disk Operating System). You can use the following three data formats in a transfer: ASCII, image, and packed image. ASCII files conform to the American Standard Code for Information Interchange in which each character is represented by a 7–bit code. In ASCII mode, FILEX deletes null and rubout characters, as well as parity bits.

NOTE

If you attempt to use RT–11 volumes for both input and output, FILEX generates an error message.

Because the file structure and data formats for each system vary, options are needed in the command line to indicate the file structures and the data formats involved in the transfer. These options are discussed in Section 14.2.1. FILEX assumes that all devices are RT–11-structured. You can use options to indicate otherwise.

14.2 Calling and Using FILEX

To call FILEX from the system device, respond to the dot printed by the keyboard monitor by typing:

```
R FILEX <RET>
```

The Command String Interpreter prints an asterisk at the left margin of the terminal and waits for you to enter a command.

Type two CTRL/C's to halt FILEX at any time (or a single CTRL/C to halt FILEX when it is waiting for console terminal input) and return control to the monitor. To restart FILEX, type R FILEX or REENTER in response to the monitor's dot.

14.3 Options

Table 14–2 lists the options that initiate various FILEX operations. The table divides into three categories: transfer, operation, and device. Transfer options direct FILEX to copy data in a certain mode. The three transfer modes are: ASCII, image, and packed image.

Operation options perform another function in addition to the data transfer. These additional functions include deleting files, producing listings, and zeroing device directories. Device options indicate the formats of devices that are involved in a transfer. These formats are DOS/BATCH or RSTS, DECsystem–10, and interchange. FILEX accepts one transfer option and one operation option in a single command.

You can specify one device option for each file involved in the transfer. The device options (/S, /T and /U) must appear following the device and file name to which they apply; other options may appear anywhere in the command

line. These options are explained in more detail in the sections following Table 14–2.

14.3.1 Transferring Files Between RT–11 and DOS/BATCH (or RSTS)

You can transfer files between block-replaceable devices used by RT–11 and the PDP–11 DOS/BATCH system. Input from DOS/BATCH may be either disk or DECtape. You can use both linked and contiguous files.

If the input device is a DOS/BATCH disk, you should specify a DOS/BATCH user identification code (UIC) in the form [nnn,nnn], where *nnn* represents an octal integer less than or equal to 377. The first part of the code represents a user group number; the second is the individual user number. The initial default value is [1,1]. The UIC you supply will be the default for all future transfers. If you do not specify a UIC, FILEX will use the current default UIC. Note that the square brackets ([ ]) are part of the UIC; you must type them if you specify a UIC.

Output to DOS/BATCH is limited to DECtape only. You do not need a UIC in a command line where you are accessing only DECtape. Individual users do not "own" files on DECtape under DOS. However, no error occurs if you do use a UIC. DECtape used under the RSTS system is legal as both input and output, since its format is identical to DOS/BATCH DECtape. You may use any valid RT–11 file storage device for either input or output in the transfer. The RT–11 device DK: is assumed if you do not indicate a device.

An RT–11 DECtape can hold more information than a DOS/BATCH or RSTS DECtape. When you copy files from a full RT–11 tape to a DOS DECtape, some information may not transfer. In this case, an error message prints and the transfer does not complete.

When a transfer from an RT–11 device to a DOS DECtape occurs, the block size of the file can increase. However, if the file is later transferred back to an RT–11 device, the block size does not decrease.

Table 14–2: FILEX Options

Options	Explanation
Transfer	
/A	Indicates a character-by-character ASCII transfer in which FILEX deletes rubouts and nulls. If you use /U with /A, FILEX also ignores all sector boundaries on the diskette. If you use /A with /T, FILEX assumes that each PDP–10 36-bit word contains five 7-bit ASCII bytes. If you use /U with /A, FILEX assumes that records are to be terminated by a line feed, vertical tab, or form feed. The transfer terminates when a CTRL/Z is encountered. (This feature is included for compatibility with RSTS.) FILEX does not transfer the CTRL/Z.

(continued on next page)

Table 14–2: FILEX Options (Cont.)

Options	Explanation
/I	Performs an image mode transfer. If the input is DOS/BATCH, RSTS, or RT–11, the transfer is word-for-word. If the input is from DECsystem–10, /I indicates that the file resembles a file created on DECsystem–10 by MACY11, MACX11, or LNKX11 with the /I option. In this case, each PDP–10 36-bit word will contain one PDP–11 8-bit byte in its low-order bits. If input or output is an interchange diskette, FILEX reads and writes four diskette sectors for each RT–11 block.
/P	Performs a packed image mode transfer. If the input is DOS/BATCH, RSTS, or RT–11, the transfer will be word-for-word. If the input is from DECsystem-10, /P indicates that the file resembles a file created on DECsystem–10 by MACY11, MACX11, or LNKX11 with the /P option. In this case, each PDP–10 36-bit word will contain four PDP–11 8-bit bytes aligned on bits 0, 8, 18, and 26. This is the default mode. If the output is interchange diskette, FILEX writes the data as EBCDIC.
Operation	
/D	Deletes the file you specify from the device directory. This option is valid only for DOS/BATCH, RSTS DECtape, and interchange diskette.
/F	Produces a brief listing of the device directory on the terminal. It lists only file names and file types. FILEX can only list directories of block-replaceable devices, and those directories only on the console terminal.
/L	Produces a complete listing of the device directory on the console terminal, including file names, block lengths, and creation dates.
/Y	Suppresses the *dev:/Init are you sure?* message
/Z	Initializes the directory of the device you specify. This option is valid only for DOS/BATCH, RSTS DECtape, and interchange diskette.
Device	
/S	Indicates that the device is a legal DOS/BATCH or RSTS block-replaceable device.
/T	Indicates that the device is a legal DECsystem–10 DECtape.
/U[:n.]	Indicates that the device is an interchange diskette. The symbol *n.* represents the length of each output record, in characters; *n.* is a decimal integer in the range 1–128. The default value is 80; *n.* is not valid with an input file specification, or with /A or /I.

To transfer a file from a DOS/BATCH block-replaceable device or RSTS DECtape to an RT–11 device, use this command syntax:

*output-filespec = input-filespec/S[/option]

where:

output-filespec — represents any valid RT–11 device, file name, and file type (if the device is not file structured, you may omit the file name and file type)

input-filespec — represents the DOS/BATCH or RSTS device ,UIC, file name, and file type to be transferred (See Table 14–1 for a list of valid devices.)

/S	is the option that designates a DOS/BATCH or RSTS block-replaceable device (This option must be included in the command line.)
/option	is one of the three transfer options from Table 14–2

To transfer files from an RT–11 storage device to a DOS/BATCH or RSTS DECtape, use this command syntax:

*DTn:output-filename/S[/option] = input-filespec

where:

DTn:output-filename	represents the file name and file type of the file to be created, as well as the DOS/BATCH or RSTS DECtape on which to store the file
input-filespec	represents the device, file name, and file type of the RT–11 file to be transferred
/S	is the option that designates a DOS/BATCH or RSTS DECtape (This option must be included in the command line.)
/option	is one of the three transfer options from Table 14–2

The following examples illustrate the use of the /S option.

The following command instructs FILEX to transfer a file called SORT.ABC from the RT–11 default device DK: to a DECtape in DOS/BATCH or RSTS format on unit DT2. The transfer is in image mode.

```
*DT2:SORT.ABC/S=SORT.ABC/I
```

The next command allows a file to be transferred from DOS/BATCH (or RSTS) DECtape to the line printer under RT–11. The transfer is done in ASCII mode.

```
*LP:=DT2:FIL.TYP/S/A
```

The next command causes the file MACR1.MAC to be transferred from the DOS/BATCH disk on unit 1, stored under the UIC [1,2], to the RT–11 device DK:. [1,2] becomes the default UIC for any further DOS/BATCH operations.

```
*DK:*.*=RK1:[1,2]MACR1.MAC/S
```

14.3.2 Transferring Files Between RT–11 and Interchange Diskette

You can transfer files between block-replaceable devices used by RT–11 and interchange format diskettes. Files are transferred in one of three formats: ASCII, image, and packed image EBCDIC mode.

A universal diskette consists of 77 tracks (some of which are reserved), each containing 26 sectors numbered from 1 to 26. A sector contains one record of

128 or fewer characters. A record must begin on a sector boundary on an interchange diskette in packed image mode. There must only be one record per sector. If a record does not fill a sector, the remainder is filled with blanks. Since packed image EBCDIC mode is inefficient and wastes space, it is recommended only to read or write diskettes that must be compatible with IBM 3741 format. Packed image (EBCDIC) mode is generally compatible with IBM 3741 format. (FILEX does not support error mapping of bad sectors and multi-volume files.) Packed image (EBCDIC) is the default mode, so you must use one of the options from Table 14–2 to specify ASCII or image mode. All records of a file must be the same size. You indicate this with the /U:n. option.

NOTE

File types are not normally recognized in interchange format; instead, a single, eight-character file name is used. However, in order to provide uniformity throughout RT–11, FILEX has been designed to accept a six-character file name with a two-character file type. If you transfer a file from RT–11 to interchange diskette, any three-character file type is truncated to two characters.

To transfer files from RT–11 format to interchange format, use this command syntax:

*output-filespec/U[:n.][/option] = input-filespec

where:

output-filespec	represents the device, file name, and file type of the interchange file to be created. Note that you cannot use wildcards in the output file specifications
/U[:n.]	is the option that designates an interchange diskette. This option must be included in the command line; *n.* represents the length of each output record, in characters; 1≤n≤128 (default is 80)
/option	is one of the three transfer options from Table 14–2
input-filespec	represents the device, file name, and file type of the RT–11 file to be transferred

To transfer files from interchange diskette to RT–11 format, use this command syntax:

*output-filespec = input-filespec/U[/option]

where:

output-filespec	represents the device, file name, and file type of the RT–11 file to be created. Note that you can use wildcards as input

input-filespec	represents the device, file name, and file type of the interchange file to be transferred
/U	is the option that designates an interchange diskette (This option must be included in the command line.)
/option	is one of the three transfer options from Table 14–2

The following command transfers the file IVAN.CAT from RT–11 RK05 unit 2 to the diskette on unit 1. The transfer is done in exact image mode (indicated by /I), ignoring all sector boundaries.

```
*DX1:IVAN.CAT/U/I=RK2:IVAN.CAT
```

The next command instructs FILEX to transfer the file BENMAR.FRM from the RT–11 disk unit 2 to the diskette on unit 0, and rename it KENJOS.JO. The /U option indicates that the format is to be changed from ASCII to the interchange format. There will be one record per sector of 128 or fewer characters. If there are fewer than 128 characters, the remainder of the sector will be filled with spaces.

```
*DY0:KENJOS.JO/U=RK2:BENMAR.FRM
```

The next command transfers the file TYPE.SET from RT–11 diskette unit 0 to the interchange diskette on unit 2. The exchange converts ASCII to interchange format, putting a maximum of 7 (indicated by :7.) characters into each sector until the entire record has been transferred. Records in excess of seven characters will be broken up and placed in succeeding sectors on the diskette. New records always begin on a sector boundary; carriage returns and line feeds are discarded. However, if you use /A or /I, FILEX ignores boundary limits and preserves carriage returns and line feeds.

```
*DX2:TYPE.SE/U:7.=RX0:TYPE.SET
```

File TYPE.SET before transfer:

ABCDEFGHIJKLMN

File TYPE.SET after transfer:

ABCDEFG—(spaces up to 128 characters) Sector 1
HIJKLMN—(spaces up to 128 characters) Sector 2

The next command copies file IVAN.CA from the interchange diskette on unit 1 to the RT–11 line printer, treating the input as ASCII characters. Note that once a record has been divided into sectors, it cannot be transferred back to its original size.

```
*LP:=DX1:IVAN.CA/U/A
```

14.3.3 Transferring Files to RT–11 from DECsystem–10

Output may be to any valid RT–11 device. DECsystem–10 DECtape is the only valid input device. To transfer files from DECsystem–10 format to RT–11 format, use this command syntax:

*output-filespec = input-filespec/T[/option]

where:

output-filespec	represents any valid RT–11 device, file name, and file type (If the device is not file-structured, you can omit the file name and file type.)
input-filespec	represents the DECtape unit, file name, and file type of the DECsystem–10 file to be transferred
/T	is the option that signifies a DECsystem–10 DECtape (When you use /T, and especially when you also use /A, the system clock loses time. Examine the time, and reset it if necessary with the TIME command.)
/option	is one of the three transfer options from Table 14–2

You cannot convert RT–11 files to DECsystem–10 format directly. However, there is a two-step procedure for doing this. First, run RT–11 FILEX and convert the files to DOS formatted DECtape. Then run DECsystem–10 FILEX to read the DOS DECtape.

The following command converts the ASCII file STAND.LIS from DECsystem–10 ASCII format to RT–11 ASCII format and stored under RT–11 on DECtape unit 2 as STAND.LIS.

```
*DT2:STAND/LIS=DT1:STAND.LIS/T/A
```

Transfers from DECsystem–10 DECtape to RT–11 DECtape may cause an <UNUSED> block to appear after the file on the RT–11 device. This is a result of the way RT–11 handles the increased amount of information on a DECsystem–10 DECtape.

The next command indicates that all files on DECsystem–10 DECtape 0 with the file type .LIS are to be transferred to the RT–11 system device using the same file name and a file type of .NEW. The /P option is the assumed transfer mode.

```
*SY:*.NEW=DT0:*.LIS/T
```

Files may not be transferred to RT–11 devices from a DECsystem–10 DECtape if a foreground job is running. This restriction is due to the fact that when FILES READS DECsystem–10 files, it accesses the DECtape control registers directly instead of using the RT–11 DECtape control handler.

14.3.4 Listing Directories

You can list at the terminal a directory of any of the block-replaceable devices used in a FILEX transfer. The command syntax is:

*device:/L/option

where:

device represents the block-replaceable device. These are the valid device types:

DOS/BATCH, RSTS DTn:, RKn:

DECsystem–10 DTn:

interchange diskette DXn: DYn:

/L is the listing option (You can substitute /F if you want a brief listing of file names only.)

/option is /S, /T, or /U. These are the valid format and option combinations:

DOS/BATCH, RSTS /S

DECsystem–10 /T

interchange diskette /U

The following example shows the complete disk directory for UIC[1,7] of the device RK1:. The letter C following the file size on a DOS/BATCH or RSTS directory listing indicates that the file is contiguous.

```
*RK1:/L/S
BADB      .SYS      1        22-JUL-74
MONLIB    .CIL    175C       22-JUL-74
DU11      .PAL     45        24-JUL-74
VERIFY    .LDA     67C       22-JUL-74
CILUS     .LDA     39        22-JUL-74
```

The next example is a command that lists all files with file type .PAL that are stored on DECtape unit 1.

```
*DT1:*.PAL/L/S
```

The next command produces a brief directory listing of the interchange diskette on unit 0, giving file names only.

```
*DX0:/U/F
```

The following command lists all files on DECsystem–10 formatted DECtape unit 1, regardless of file name or file type; with the /F, a brief directory is requested in which only file names print.

```
*DT1:*.*/F/T
```

14.3.5 Deleting Files from DOS/BATCH (RSTS) DECtapes and Interchange Diskettes

Use FILEX to delete files from DOS/BATCH and RSTS formatted DECtapes, and from interchange diskettes.

To delete files, use this command syntax:

*filespec/D/option

where:

filespec	represents the device, file name, and file type of the file to be deleted
/D	is the delete option
/option	can be either /S, for DOS/BATCH and RSTS block-replaceable devices, or /U, for interchange diskettes

The following command deletes all files with the file type .PAL on DECtape unit 0.

```
*DT0:*.PAL/D/S
```

The next command deletes the file TABLE.OBJ from the DECtape on unit 2.

```
*DT2:TABLE.OBJ/D/S
```

The next command deletes all files with an .RNO file type from the interchange diskette on unit 0.

```
*DX0:*.RN/D/U
```

You can also use FILEX to initialize the directories of DOS/BATCH and RSTS DECtapes, and interchange diskettes. Use this command syntax:

*device:/Z/option[/Y]

where:

device	represents the DOS/BATCH or RSTS DECtape, or the interchange diskette to be zeroed
/Z	is the initialize option
/option	can be either /S, for DOS/BATCH and RSTS DECtapes, or /U, for interchange diskettes
/Y	inhibits the FILEX verification message

The following command directs FILEX to initialize the directory of the interchange diskette on unit 0.

```
*DX0:/Z/U
```

FILEX prints a verification message:

```
DX0:/Init are you sure?
```

Respond with a Y for initialization to begin. Any other response aborts the command.

The next command initializes the DECtape on unit 1 in DOS/BATCH (RSTS) format. Note that by using the /Y option you suppress the verification message.

```
*DT1:/Z/S/Y
```

NOTE

The directory of an initialized interchange diskette has a single file entry, DATA, that reserves the entire diskette. You must delete this file before you can write any new files on the diskette. This is necessary for IBM compatibility. Do this by using the following command:

```
*DX0:DATA/D/U
```

Chapter 15
Source Compare (SRCCOM)

The RT–11 source compare program (SRCCOM) compares two ASCII files and lists the differences between them. SRCCOM can either print the results or store them in a file. SRCCOM is particularly useful when you want to compare two similar versions of a source program. A file comparison listing highlights the changes made to a program during an editing session.

SRCCOM is also useful for creating a command file that you can run with the source language patch program (SLP), described in Chapter 24. When you use SRCCOM for creating a command file, you can patch one version of a source file so that it matches another version. Section 15.4 describes how to create a command file for SLP.

15.1 Calling and Using SRCCOM

To call SRCCOM from the system device, respond to the dot (.) printed by the keyboard monitor by typing:

```
R SRCCOM <RET>
```

The Command String Interpreter prints an asterisk at the left margin of the terminal and waits for you to enter a command string. If you respond to the asterisk by entering only a carriage return, SRCCOM prints its current version number. The syntax of the command is:

[output-filespec=]old-filespec,new-filespec[/option...]

where:

output-filespec	represents the destination device or file for the listing of differences
old-filespec	represents the first file to be compared
new-filespec	represents the second file to be compared
option	is one of the options listed in Table 15–1

Note that you can specify the input files in any order if you want only a comparison. If you are creating a patch for use with the SLP utility, then specify the input files in the "old-file, new-file" order shown above. The console terminal is the default output device. The default file type for input files is .MAC. SRCCOM assigns .DIF as the default file type for output files.

You can type CTRL/C to halt SRCCOM and return control to the monitor when SRCCOM is waiting for input from the console terminal. You must

type two CTRL/Cs to abort SRCCOM at any other time. To restart SRCCOM, type R SRCCOM or REENTER and a carriage return in response to the monitor's dot.

SRCCOM examines the two source files line by line, looking for groups of lines that match. When SRCCOM finds a mismatch, it lists the lines from each file that are different. SRCCOM continues to list the differences until a specific number of lines from the first file match the second file. The specific number of lines that constitutes a match is a variable that you can set with the /L:n option.

15.2 Options

Table 15–1 summarizes the operations you can perform with SRCCOM options. You can place these options anywhere in the command string, but it is conventional to place them at the end of the command line.

Table 15–1: SRCCOM Options

Option	Explanation
/A	Lets you specify an audit trail (a string of characters that marks each updated line of a patched source file). Use /A with the /P option. The file you create with these options can be used as command file input for the source language patch program SLP (see Chapter 24).
/B	Compares blank lines; normally, SRCCOM ignores blank lines.
/C	Ignores comments (all text on a line preceded by a semicolon) and spacing (spaces and tabs). A line consisting entirely of a comment is still included in the line count.
/D	Creates a listing of the "new file" specified in the command line with the differences from the "old file" marked with vertical bars (¦) to indicate insertions and bullets (o) to indicate deletions.
/F	Includes form feeds in the output listing; SRCCOM normally compares form feeds, but does not include them in the output listing.
/L[:n]	Specifies the number of lines that determines a match; *n* is an octal integer in the range 1 through 310. The default value for *n* is 3.
/P	Creates for output a file that can be an input command file for the source language patch program SLP (see Chapter 24).
/S	Ignores spaces and tabs.
/T	Compares blanks and tabs that appear at the end of a line. Normally SRCCOM ignores these trailing blanks and tabs.
/V:i:d	Use with /D to specify the characters you want SRCCOM to use in place of vertical bars and bullets. This option is useful if your terminal does not print the vertical bar character. Both *i* and *d* represent the numeric codes for ASCII characters in the range 40 through 176 (octal), where *i* represents the code for the insertion character, and *d* the deletion character code.

15.3 Output Format

This section describes the SRCCOM output listing format and explains how to interpret it.

15.3.1 Sample Text

It will be helpful first to look at a sample text file, DEMO.BAK:

```
        FILE1
HERE'S A BOTTLE AND AN HONEST FRIEND!
  WHAT WAD YE WISH FOR MAIR, MAN?
WHA KENS, BEFORE HIS LIFE MAY END,
  WHAT HIS SHAME MAY BE O' CARE, MAN?
THEN CATCH THE MOMENTS AS THEY FLY,
  AND USE THEM AS YE OUGHT, MAN!--
BELIEVE ME, HAPPINESS IS SLY,
  AND COMES NOT AYE WHEN SOUGHT, MAN.

                              --SCOTTISH SONG
```

This file contains two typing errors. In the fourth line of the song, *shame* should be *share*. In the seventh line, *sly* should be *shy*. Here is a file called DEMO.TXT that has the correct text:

```
        FILE2
HERE'S A BOTTLE AND AN HONEST FRIEND!
  WHAT WAD YE WISH FOR MAIR, MAN?
WHA KENS, BEFORE HIS LIFE MAY END,
  WHAT HIS SHARE MAY BE O' CARE, MAN?
THEN CATCH THE MOMENTS AS THEY FLY,
  AND USE THEM AS YE OUGHT, MAN!--
BELIEVE ME, HAPPINESS IS SHY,
  AND COMES NOT AYE WHEN SOUGHT, MAN.

                              --SCOTTISH SONG
```

15.3.2 Sample Output Listing

SRCCOM can list the differences between the two files. The example below compares the original file, DEMO.BAK, to its edited version, DEMO.TXT:

```
*DEMO.BAK,DEMO.TXT/L:1
1) DK:DEMO.BAK
2) DK:DEMO.TXT
**********
1)1              FILE1
1)      HERE'S A BOTTLE AND AN HONEST FRIEND!
****
2)1              FILE2
2)      HERE'S A BOTTLE AND AN HONEST FRIEND!
```

(continued on next page)

```
**********
1)1        WHAT HIS SHAME MAY BE O' CARE, MAN?
1)       THEN CATCH THE MOMENTS AS THEY FLY,
****
2)1        WHAT HIS SHARE MAY BE O' CARE, MAN?
2)       THEN CATCH THE MOMENTS AS THEY FLY,
**********
1)1      BELIEVE ME, HAPPINESS IS SLY,
1)         AND COMES NOT AYE WHEN SOUGHT, MAN.
****
2)1      BELIEVE ME, HAPPINESS IS SHY,
2)         AND COMES NOT AYE WHEN SOUGHT, MAN.
**********
?SRCCOM-W-Files are different
```

SRCCOM always prints the file name of each file as identification:

```
1) DK:DEMO.BAK
2) DK:DEMO.TXT
```

The numbers at the left margin have the form *n)m*, where *n* represents the source file (either 1 or 2) and *m* represents the page of that file on which the specific line is located.

SRCCOM next prints ten asterisks and then lists the differences between the two files. The /L:n option was used in this example to set to 1 the number of lines that must agree to constitute a match.

The first line of both files differs. SRCCOM prints the first line from the first file, followed by the second line as a reference. SRCCOM then prints four asterisks, followed by the corresponding two lines of the second file.

```
1)1              FILE1
1)      HERE'S A BOTTLE AND AN HONEST FRIEND!
****
2)1              FILE2
2)      HERE'S A BOTTLE AND AN HONEST FRIEND!
**********
```

The fourth line contains the second discrepancy. SRCCOM prints the fourth line from the first file, followed by the next matching line as a reference.

```
1)1        WHAT HIS SHAME MAY BE O' CARE, MAN?
1)       THEN CATCH THE MOMENTS AS THEY FLY,
****
```

The four asterisks terminate the differences section from the first file.

SRCCOM then prints the fourth line from the second file, again followed by the next matching line as a reference:

```
2)1        WHAT HIS SHARE MAY BE O' CARE, MAN?
2)       THEN CATCH THE MOMENTS AS THEY FLY,
**********
```

The ten asterisks terminate the listing for a particular difference section.

SRCCOM scans the remaining lines in the files in the same manner. When it reaches the end of each file, it prints the *?SRCCOM-W-Files are different* message on the terminal.

The second example is slightly different. The default value for the /L:n option sets to 3 the number of lines that must agree to constitute a match. The output listing is directed to the file DIFF.TXT on device DK:.

```
*DIFF.TXT=DEMO.BAK,DEMO.TXT
?SRCCOM-W-Files are different
```

The monitor TYPE command lists the information contained in the output file:

```
.TYPE DIFF.TXT
1) DK:DEMO.BAK
2) DK:DEMO.TXT
**********
1)1                FILE1
1)       HERE'S A BOTTLE AND AN HONEST FRIEND!
****
2)1                FILE2
2)       HERE'S A BOTTLE AND AN HONEST FRIEND!
**********
1)1          WHAT HIS SHAME MAY BE O' CARE, MAN?
1)       THEN CATCH THE MOMENTS AS THEY FLY,
1)          AND USE THEM AS YE OUGHT, MAN:--
1)       BELIEVE ME, HAPPINESS IS SLY,
1)          AND COMES NOT AYE WHEN SOUGHT, MAN.
****
2)1          WHAT HIS SHARE MAY BE O' CARE, MAN?
2)       THEN CATCH THE MOMENTS AS THEY FLY,
2)          AND USE THEM AS YE OUGHT, MAN:--
2)       BELIEVE ME, HAPPINESS IS SHY,
2)          AND COMES NOT AYE WHEN SOUGHT, MAN.
**********
```

As in the first example, SRCCOM prints the file name of each file:

```
1) DK:DEMO.BAK
2) DK:DEMO.TXT
```

The first line of both files differs, so SRCCOM prints the first two lines of both files, as in the listing at the terminal from the previous example:

```
1)1             FILE1
1)      HERE'S A BOTTLE AND AN HONEST FRIEND!
****
2)1             FILE2
2)      HERE'S A BOTTLE AND AN HONEST FRIEND!
**********
```

Again, the fourth line differs. SRCCOM prints the fourth line of the first file, followed by the next matching line:

```
1)1         WHAT HIS SHAME MAY BE O' CARE, MAN?
1)        THEN CATCH THE MOMENTS AS THEY FLY,
```

However, SRCCOM did not find a match (three identical lines) before it encountered the next difference. So, the second matching line prints, followed by the next differing line from the first file:

```
1)          AND USE THEM AS YE OUGHT, MAN:--
1)        BELIEVE ME, HAPPINESS IS SLY,
```

Again, the next matching line prints:

```
1)          AND COMES NOT AY WHEN SOUGHT, MAN.
```

The /B option to include blank lines in the comparison was not used in this example. Thus, SRCCOM recognizes only one more line before the end of file. Since the two identical lines do not constitute a match (three are needed), SRCCOM prints the last line as part of the difference section for the first file:

```
1)
1)                          --SCOTTISH SONG
1)
****
```

In a similar manner, SRCCOM prints a differences section for the second file, ending the listing with the *?SRCCOM-W-Files are different* message.

NOTE

Regardless of the output specification, the differences message always prints on the terminal. If you compare two files that are identical and specify a file for the output listing, the message *?SRCCOM-I-No differences found* prints on the terminal and SRCCOM does not create an output file.

15.3.3 Changebar Option /D[/V:i:d]

When you use the /D option in the SRCCOM command line, SRCCOM creates a listing in which it inserts vertical bars (¦) and bullets (o) to denote the differences between the two files in the command line. The vertical bar indicates insertion; the bullet indicates deletion. If you do not specify an output file SRCCOM prints the listing at the terminal.

If you include the /V:i:d option with /D (you cannot use /V:i:d without /D), you can specify what characters you would like in place of the vertical bar and/or bullet. The argument *i* represents the ASCII code (between 40 and 176 (octal)) for the character you want in place of the vertical bar. The argu-

ment *d* represents the ASCII code (between 40 and 176) for the character you want to use in place of the bullet.

In the following command line, SRCCOM compares DEMO.BAK to DEMO.TXT:

```
*DEMO.BAK,DEMO.TXT/D/L:1
```

When SRCCOM processes the last command, it prints at the terminal the following listing:

```
!              FILE2
     HERE'S A BOTTLE AND AN HONEST FRIEND!
       WHAT WAD YE WISH FOR MAIR, MAN?
     WHA KENS, BEFORE HIS LIFE MAY END,
!      WHAT HIS SHARE MAY BE O' CARE, MAN?
     THEN CATCH THE MOMENTS AS THEY FLY,
       AND USE THEM AS YE OUGHT, MAN:
!    BELIEVE ME, HAPPINESS IS SHY,
       AND COMES NOT AYE WHEN SOUGHT, MAN.

                                    --SCOTTISH SONG
?SRCCOM-W-Files are different
```

15.4 Creating a SLP Command File

You can use SRCCOM to create an input command file to the source language patch program SLP described in Chapter 24. Two SRCCOM options, /A and /P, are designed solely for creating this command file.

15.4.1 Patch Option (/P)

When you specify an output file and include the /P option in the SRCCOM command line, SRCCOM creates a file you can use as the SLP input command file. In the following sample command line, SRCCOM creates an output file, PATCH.DIF, which contains the necessary commands that, when used with SLP, can patch DEMO.BAK so that it matches DEMO.TXT.

```
*PATCH=DEMO.BAK,DEMO.TXT/P
```

15.4.2 Audit Trail Option (/A)

You can use the /A option with /P to specify an audit trail. When SLP patches a file, it creates two output files. One output file is the patched source file; the second is a listing file. The listing file contains a numbered listing of the patched file, and it also has an audit trail SLP has appended to each changed line. The audit trail is a string of characters that keeps track of the update status of each line in the patched source file. SLP appends the audit trail to the right margin of each updated line in the patched source file. Note that when SLP must change a line, it appends an additional audit

trail below the audit trail of the changed line. The additional audit trail keeps track of the number of consecutive lines that change.

When you use the /A option with /P you can specify what characters you want in the audit trail.

When you use the /A option, SRCCOM prompts you for the audit trail:

```
Audit trail?
```

Respond with a string of up to 12 characters. Do not use a slash (/) in the audit trail. The example below provides a sample of a SLP listing file that contains a meaningful audit trail: the author's initials and the date-of-patch

```
SLP V04.00                    ADDRSS,ADDRSS=ADDRSS,ADDRSS

   1.  FOURSCORE AND SEVEN YEARS AGO,
   2.  OUR FATHERS BROUGHT FORTH ON THIS CONTINENT
   3.  A NEW NATION, CONCEIVED IN LIBERTY                ;AL-4*29*1863
   4.  AND DEDICATED TO THE PROPOSITION                  ;AL-4*29*1863
   5.  THAT ALL MEN ARE CREATED EQUAL.                   ;**-2
   6.  NOW WE ARE ENGAGED IN A GREAT CIVIL WAR,
   7.  TESTING WHETHER THAT NATION, OR ANY NATION        ;AL-4*29*1863
   8.  SO CONCEIVED AND SO DEDICATED,                    ;**-1
   9.  CAN LONG ENDURE.
```

Chapter 16
BINCOM

The RT–11 binary compare program (BINCOM) compares two binary files and lists the differences between them. BINCOM can either print the results at the terminal or line printer, or store them in a file. It is particularly useful when you need to compare two executable programs because it provides a quick way of telling whether two data files are identical. Another use of BINCOM is to verify whether two versions of a program produce identical output files when given identical input files.

You can also use BINCOM to create an indirect command file that invokes the save image patch program (SIPP, described in Chapter 22) to patch one version of a file so that it matches another version. Section 16.5 describes the procedure you can use to create an indirect command file for SIPP.

16.1 Calling and Using BINCOM

To call BINCOM from the system device, respond to the dot (.) printed by the keyboard monitor by typing:

```
R BINCOM <RET>
```

The Command String Interpreter prints an asterisk at the left margin of the terminal and waits for you to enter a command string. If you respond to the asterisk by entering only a carriage return, BINCOM prints its current version number. You can type a CTRL/C to halt BINCOM and return control to the monitor when BINCOM is waiting for input from the console terminal. You must type two CTRL/Cs to abort BINCOM at any other time. To restart BINCOM, type R BINCOM or REENTER and a carriage return in response to the monitor's dot. The syntax of the command is:

[output-spec[/option]][,patch-spec[/option]] = input-spec1,
input-spec2[/option...]

where:

output-spec	represents the file or volume that you want the differences between the two files you are comparing to be sent to. If omitted, default is TT:
patch-spec	represents the file that you can run as an indirect command file; it contains the commands necessary to patch *input-spec1* so it matches *input-spec2*
input-spec1	represents the first file to be compared

input-spec2 represents the second file to be compared

option is one or more of the options listed in Table 16–1

The console terminal is the default output device. There is no default file type for input files; you must always specify the file type. BINCOM assigns .DIF as the default file type for the difference output file, and .COM as the default file type for the SIPP indirect command file.

BINCOM examines the two input files word by word (or byte by byte), looking for differences. When BINCOM finds a mismatch, it prints the block number and offset within the block at which the difference occurs, the octal values from the first and second input files, and the logical exclusive OR of the two values. This last number helps you find the bits that are different in the two values.

16.2 Options

Table 16–1 summarizes the options that you can use with BINCOM. Except for the /O option, you can place these options anywhere in the command string, but it is conventional to place them at the end of the command line.

Table 16–1: BINCOM Options

Option	Explanation
/B	Compares the input files byte by byte. If you do not specify this option, BINCOM compares the files word by word.
/E:n	Ends comparison at block number *n*, where *n* is an octal value. If you do not include this option, BINCOM ends the comparison when it reaches end-of-file on one of the input files.
/H	Types on the console terminal the list of available options.
/O	Creates an output file or patch file, even if there are no differences between the two input files. If you enter this option after the differences output file, BINCOM creates the differences output file whether or not there are differences between the two input files. If you enter this option after the SIPP indirect command file, BINCOM creates an SIPP indirect command file whether or not differences exist. You can enter this option at the end of the command line if you want both output files. This option is useful in BATCH streams to prevent later job steps from failing because BINCOM did not create the expected control file.
/Q	Suppresses the printing of the differences and prints only the message: *?BINCOM-W-Files are different*, if applicable (or *?SIPP-I-No differences encountered*). This option is useful in BATCH control files when you want to test for differences and perhaps abort execution, but do not want the log file filled with output.
/S:n	Starts the comparison at block number *n*, where *n* is an octal value.

16.3 Output Format

This section describes the BINCOM output file format and explains how to interpret it.

If you include an output file specification in the command line, BINCOM creates a file that contains the differences between the two input files. If you do not specify an output file, BINCOM prints the differences only on the terminal. If you include the /Q option, BINCOM does not print the differences and does not create an output file.

The first line of the difference listing is a header line that identifies the files you are comparing. Next, BINCOM prints a blank line and then lists the differences between the two files. Each difference line has the following format:

```
bbbbbb ooo/ ffffff ssssss xxxxxx
```

where:

bbbbbb	is the octal number of the block that contains the difference
ooo	is the octal offset within the block
ffffff	is the value in the first file
ssssss	is the value in the second file
xxxxxx	is the logical exclusive OR of the two values

If there are several differences in a block, BINCOM prints the block number only once for that block. Thus, each time you see a block number appear, it indicates that the differences being printed are in a new block.

If you specify the /B option to compare the files byte by byte, BINCOM prints the numbers *ffffff*, *ssssss*, and *xxxxxx* as three-digit, octal byte values.

When BINCOM reaches the end of one of the input files, it checks its position in the other input file. If the files have different lengths, BINCOM prints the message:

```
?BINCOM-W-File is longer -- dev:filnam.typ
```

When BINCOM reaches the end of both files, it prints on the terminal the message if it encountered any difference:

```
?BINCOM-W-Files are different
```

If the two files are identical, BINCOM prints the message:

```
?BINCOM-I-No differences encountered
```

If you include a SIPP indirect command file specification in the command line, BINCOM creates a file that is a valid command file for the save image patch program SIPP (see Chapter 22). This command file contains commands that instruct SIPP to patch the first input file so that it matches the second input file. If you want BINCOM to create only the patch file, enter a

comma before the patch file specification in the command line, in place of the output file specification.

16.4 Examples

The first example compares files TEST1.TST and TEST3.TST, both on device DK:. Notice that there are no output files and no options in the command line.

```
.R BINCOM
*TEST1.TST,TEST3.TST
BINCOM comparing/DK:TEST1.TST -- DK:TEST3.TST

000000  002/   051511  051502  000013
?BINCOM-W-Files are different
```

Notice the fourth line in the above example. The third number, 051511, represents the contents of location 2, block 0, in file TEST1.TST. The fourth number, 051502, represents the contents of the same location in file TEST3.TST. The last number is the logical exclusive OR of the two values.

The next example specifies the output file FOO1 as the file in which to store the differences between TEST1.TST and TEST3.TST.

```
.R BINCOM
*FOO1=TEST1.TST,TEST3.TST
?BINCOM-W-Files are different
```

The contents of file FOO1 from the last example follow. Note that FOO1 has the default file type .DIF.

```
.TYPE FOO1.DIF
BINCOM comparing/DK:TEST1.TST -- DK:TEST3.TST

000000  002/   051511  051502  000013
```

16.5 Creating a SIPP Command File

You can use BINCOM to create an indirect command file that invokes the save image patch program (SIPP, described in Chapter 22) to patch one version of a file you are comparing to match the other version. As noted earlier in this chapter, you specify this indirect file as the second output file in the CSI command string. If you wish to create only the indirect file as output, place a comma before the output file specification in the command line, in place of the first output file.

The example that follows specifies FOO2 as the patch output file, which contains the commands necessary to patch file TEST1.TST so it matches TEST3.TST. Notice the comma that appears before the patch file specification. This indicates that a difference output file is not requested, resulting in the printing out of all the differences at the terminal when the command is executed.

```
.R BINCOM
*,FOO2=TEST1.TST,TEST3.TST
BINCOM comparing/DK:TEST1.TST -- DK:TEST3.TST

000000  002/   051511  051502  000013
?BINCOM-W-Files are different
```

The contents of file FOO2 follow. Note that BINCOM assigns to this file the .COM file type.

```
.TYPE FOO2.COM
R SIPP
DK:TEST1.TST/A
000000
000000002
051502
^Y
^C
```

The file FOO2 from the previous example can be run as an indirect command file to make TEST1.TST match TEST3.TST. This can be done with the following command, when typed in response to the keyboard monitor dot:

```
@FOO2.COM
```

Chapter 17
Resource Utility Program (RESORC)

The resource utility program lists system resource information on the terminal. You can use RESORC to display the following data about your system:

- Monitor version number
- SET options in effect
- Hardware configuration
- Currently loaded jobs
- System generation special features in effect
- Device assignments
- Status of currently active terminals (on multi-terminal systems)
- Device handler status

17.1 Calling and Using RESORC

To call RESORC from the system device, respond to the keyboard monitor dot (.) by typing:

```
R RESORC <RET>
```

The Command String Interpreter prints an asterisk (*) at the left margin of the terminal and waits for your input. At this point, enter the RESORC option or options required to obtain the information you need. Section 17.2 describes these options, and Table 17–1 summarizes them.

If you enter only a carriage return in response to the asterisk, RESORC prints its name and current version number. To abort RESORC while it is executing, type two CTRL/Cs. Type one CTRL/C to return control to the monitor when RESORC is waiting for input (that is, when an asterisk has printed on the terminal).

17.2 Options

By specifying one or more of the options /C, /D, /H, /J, /M, /O, and /T, you choose the information that RESORC lists on the terminal. If you use two or more options, you can enter them in any order, although RESORC lists the information in the order /M, /C, /H, /O, /D, /J, /T.

RESORC offers two additional options that are equivalent to combinations of options. The /Z option has the same effect as a combination of /M, /C, /H, /J, and /O options. The /A option has the same effect as a combination of all the options except /Z.

Table 17–1: RESORC Options

Option	Display
/A	The result of a combination of all RESORC options (except /Z)
/C	The device from which you bootstrapped the system and the monitor SET options in effect
/D	A list of the device handlers, their status, and their vectors
/H	The hardware configuration
/J	Information about the currently running and loaded jobs
/L	Device assignments
/M	The monitor type, version number, and patch level
/O	The system generation special features in effect
/T	The status and options in effect for currently active terminals on multi-terminal systems
/Z	The result of a combination of /M, /C, /H, /J, and /O options

17.2.1 All Option (/A)

The /A option has the same effect as a combination of all the other RESORC options (except /Z). When you enter /A, all RESORC information is printed on the terminal in the order shown below.

- Monitor configuration (that is, the monitor type and version number; the base address of the resident monitor; the device from which the monitor was bootstrapped; what the SET options are; whether a foreground job is loaded; and the indirect file nesting depth)
- Hardware configuration
- System generation special features in effect
- Device handler status
- Device assignments
- Job status
- Status of currently active terminals

See the following sections for details on these topics.

17.2.2 Software Configuration Option (/C)

The /C option displays:

- The device from which you bootstrapped the system and the resident monitor base address
- The monitor SET options
- Whether a foreground job is loaded (if applicable)
- Indirect file nesting depth

The following example uses the /C option.

```
*/C

Booted from DL1:RT11FB
Resident Monitor base is 127444 (44836.)
USR is set SWAP
Foreground Job is loaded
TT is set NOQUIET
Indirect file abort level is ERROR
Indirect file nesting depth is 3
```

17.2.3 Device Handler Status Option (/D)

RESORC's /D option displays a list of your system's device handlers, along with their status and vectors. The /D option lists only those handlers whose special features match those of the currently booted monitor. If a handler is loaded, RESORC prints its load address. The following sample shows the table that RESORC prints when you use the /D option.

```
*/D

Device        Status       Vector

  DX       Installed       264
  DY       Not installed   264
  DD       Not installed   300 304 310 314
  PD       Not installed
  DT       Not installed   214
  RF       Not installed   204
  DS       Not installed   204
  RK       113154          220
  DL       Resident        160
  DP       Not installed   254
  DM       Installed       210
  MT       Installed       224
  MM       Not installed   224
  MS       Not installed   224
  LP       113764          200
  LS       Not installed   300 304
  PC       Installed       070 074
  CR       Not installed   230
  CT       Installed       260
  NL       Installed       000
```

In this table, the status column can list one of the following messages:

Message	Meaning
Installed	The device handler is in the system tables, but you have not loaded it in main memory (you can load it with the LOAD command).
Not installed	The device handler is not in the system tables, but you can add the handler with the INSTALL command (if there is a free slot).
-Not installed	The device handler special features do not match those of the monitor; you cannot install the handler. (The minus sign in front of any message means that you cannot install the handler.)
Resident	The device handler is permanently in memory; you cannot remove or unload it.
nnnnnn	The beginning address of a loaded handler

The last column in the /D listing identifies vectors. If the handler has multiple vectors, the /D option prints the additional vectors in this column.

17.2.4 Hardware Configuration Option (/H)

When you use the /H option, RESORC lists the processor type and then all the special hardware features in your system configuration. The processor is one of the following:

LSI 11
PDP 11/04
PDP 11/05,10
PDP 11/15,20
PDP 11/23
PDP 11/34
PDP 11/35,40
PDP 11/44
PDP 11/45,50,55
PDP 11/60
PDP 11/70

Any special hardware is chosen from the following list. (The /H option displays the features in the order they occur in the list.)

FP11 Hardware Floating Point Unit
Commercial Instruction Set (CIS)
Extended Instruction Set (EIS)
Floating Point Instruction Set (FIS)
KT11 Memory Management Unit
Parity Memory

Cache Memory
VT11, VT48, or VS60 Graphics Hardware

The next item that appears in the /H listing is the clock frequency (50 or 60 cycles), and the last is the KW11–P programmable clock (if your system has one and you are not using it as the system clock).

The following example shows the /H option.

```
*/H

PDP 11/34 Processor
Extended Instruction Set (EIS)
KT11 Memory Management Unit
Parity Memory
Cache Memory
60 Cycle System Clock
KW11-P User Programmable Clock
```

17.2.5 Loaded Jobs Option (/J)

The /J option prints information about the currently loaded jobs. For each job, RESORC displays:

- The job number and name (if you have not enabled system job support on your monitor, the foreground job name appears as FORE, and its priority level is 1 in FB or XM)
- The console the job is running on
- The priority level of the job
- The job's state (running, suspended, or done but not unloaded)
- The low and high memory limits of the job
- The start address of the job's impure area

The following example uses the /J option.

```
*/J

Job  Name  Console Level State    Low    High  Impure
16   MFUNCT   1       7   Suspend 115350 127444 114412
14   EL       0       6           132404 141452 130663
12   QUEUE    0       5           152345 163243 144231
 0   RESORC   0       0   Run     000000 113144 133374
```

17.2.6 Device Assignments Option (/L)

When you type /L in response to the CSI asterisk, RESORC displays your system's device assignments. The devices RESORC lists are those in the system tables. The listing also includes additional information about particular devices. The informational messages and their meanings follow.

(RESORC) or =RESORC	The device or unit is assigned to the background job RESORC (for FB and XM monitors only).
(FORE) or =FORE	The device or unit is assigned to the foreground job (only for FB and XM monitors that do not have system job support).
(jobname) or =jobname	The device or unit is assigned to the system or foreground job (only for FB and XM monitors that have system job support), where jobname represents the name of the foreground or system job.
(Loaded)	The handler for the device has been loaded into memory with the LOAD command.
(Resident)	The handler for the device is included in the resident monitor.
=logical-device-name(1), logical-device-name(2)... ,logical-device-name(n)	The device or unit has been assigned the indicated logical device names with the ASSIGN command.
xx free slots	The last line tells the number of unassigned (free) devices.

The following example was created under an FB monitor. It shows the status of all devices known to the system.

```
*/L
TT   (Resident)
DL   (Resident)
     DL1 = SY , DK , OBJ, SRC, BIN
     DL2 = LST, MAP
MQ   (Resident)
RK   (Loaded)
DM
DX   (Loaded)
     DX0: (FORE)
     DX1: (RESORC)
MT
CT
LP
PC
BA
NL
9 free slots
```

17.2.7 Current Monitor Option (/M)

When you use the /M option RESORC prints the type, version number, and patch level of the currently running monitor. The designation BL, SJ, FB, or

XM identifies the monitor type as base-line, single-job, foreground/background, or extended memory, respectively.

The following example uses the /M option.

```
*/M

RT-11FB (S) V04.00
```

17.2.8 Special Features Option (/O)

The special features chosen during system generation are listed on the terminal when you use the /O option. Whatever features are in effect are printed out in the same order as the following list of possible special features.

Option	Function
Device I/O time-out support	Permits device handlers to do the equivalent of a mark time without doing a .SYNCH request; DECnet applications require this support.
Error logging support	Keeps a statistical record of all I/O operations on devices that are supported by this feature; detects and stores data on any errors that occur during I/O operations.
Multi-terminal support	Permits you to use as many as 16 terminals.
Memory parity support	Causes the system to print an error message when a memory parity error occurs.
SJ time support	Configures the single-job monitor to include mark time and cancel mark time programmed requests and to support the .FORK process.
System job support	Allows you to run up to six jobs in the foreground in addition to the foreground and background jobs.

The following example shows the /O option

```
*/O

Device I/O time-out support
Error logging support
Multi-terminal support
Memory parity support
```

If there are no special features in effect, RESORC prints *NO SYSGEN options enabled.*

17.2.9 Terminal Status Option (/T)

The /T option displays information about currently running active terminals on multi-terminal systems. Therefore, if your system has only the console terminal, RESORC prints:

```
No multi-terminal support
```

Since multi-terminal support is not part of the distributed RT–11 monitors, such support is present on your system only if you have included it during system generation; that is, multi-terminal support is a special feature.

If you do have multi-terminal support, and you enter the /T option in response to RESORC's asterisk, RESORC prints a table similar to the following example:

```
*/T

Unit Owner      Type          WIDTH   TAB   CRLF   FORM   SCOPE   SPEED

   0        S-Console DL      132    No    Yes     No      No      N/A
   1        Local DZ           80    Yes   No      No      Yes    4800
   2        Local DZ           80    Yes   No      No      Yes    4800
   3        Local DZ           80    Yes   No      No      Yes    4800
   4        Remote DZ          72    No    Yes     No      No      300
```

Note that in this table, the unit number refers to the terminal; RT–11 multi-terminal support permits you to use as many as 16 terminals.

The UNIT OWNER column lists the name of the job (foreground, system, or background) to which the terminal is assigned. If the running monitor does not have system job support, RESORC prints FORE and RESORC, where applicable.

The TYPE column of this table shows (1) the type of terminal: local, remote, console, or S-console (a console shared between background, system, and foreground jobs) and (2) the type of hardware interface the terminal uses: DL or DZ.

The WIDTH column indicates the width in characters (up to 132) of terminal listing or display text.

The next four columns indicate which SET options are in effect on the terminal. If you have set TAB, the terminal can execute hardware tabs. If you have set CRLF, the terminal issues a carriage return and line feed whenever you attempt to type past the right margin. The terminal is capable of executing hardware form feeds if you have set FORM and (on graphics terminals) of echoing RUBOUT characters as backspace-space-backspace if you have set SCOPE.

The last column, SPEED, lists the terminal's baud rate. If the terminal's DL11 interfaces do not have programmable baud rates, however, the /T option prints N/A under the SPEED column to indicate that the baud rate is not alterable.

17.2.10 Summary Option (/Z)

The /Z option has the same effect as a combination of the /M, /C, /H, /J, and /O options. In other words, /Z lists the following information about your system:

- Monitor configuration
- Set options in effect on the monitor
- Hardware configuration
- System generation special features in effect

This information prints out in the order shown in the following sample.

```
*/Z

RT-11FB (S) V04.00

Booted from DL1:RT11FB
Resident Monitor base is 127444 (44836.)
USR is set SWAP
Foreground Job is loaded
TT is set NOQUIET
Indirect file abort level is ERROR
Indirect file nesting depth is 3

PDP 11/34 Processor
Extended Instruction Set (EIS)
KT11 Memory Management Unit
Parity Memory
Cache Memory
60 Cycle System Clock
KW11-P User Programmable Clock

Device I/O time-out support
Error logging support
Multi-terminal support
Memory parity support
```

Chapter 18
Volume Formatting Program (FORMAT)

The FORMAT utility program formats disks and diskettes. You can also use FORMAT to convert single-density diskettes to double-density and vice versa. FORMAT can format RX01-RX02 diskettes, RK05 disks, RP02-RP03 disks, and RK06-RK07 disks.

Formatting a volume makes that volume usable by RT–11. When you format a volume, FORMAT writes headers on each block in that volume. The header of a block contains data the device controller uses to transfer information to and from that block.

When you use FORMAT to convert a single-density diskette to double density, or vice versa, FORMAT writes media density marks on each block of the diskette. You can format a diskette only in a double-density diskette drive, DY:. If you attempt to format a diskette in a single-density diskette drive, DX:, FORMAT prints an error message.

Reformatting with the FORMAT program can also eliminate bad blocks that disks and diskettes sometimes develop as a result of age and use. Although formatting does not guarantee that each bad block will be eliminated, formatting can reduce the number of bad blocks.

NOTE

FORMAT destroys any data that currently exists on the disk.

18.1 Calling and Using FORMAT

To call FORMAT from the system device, respond to the dot (.) printed by the keyboard monitor by typing:

```
R FORMAT <RET>
```

The Command String Interpreter prints an asterisk (*) at the left margin of the terminal and waits for a command string. If you enter only a carriage return in response to the asterisk, FORMAT prints its current version number. You can type CTRL/C to halt FORMAT and return control to the monitor when FORMAT is waiting for input from the console terminal. You cannot type two CTRL/Cs to halt FORMAT during an operation.

If you interrupt the program during a formatting operation by some other means, the disk or diskette involved is not completely formatted. You must restart the operation on the same disk or diskette and allow it to run to completion.

Chapter 6, Command String Interpreter, describes the general syntax of the command line that system utility programs accept. FORMAT accepts one device specification (either a physical or logical device name) followed, if necessary, by one or more options. An RK05 disk you wish to format can be located in any unit (0–7) of device RK:. A diskette you need to format must be mounted on an RX02 device (device DY:), but it can be located in any unit (0–3) of that device. You cannot format diskettes on an RX01 device.

FORMAT automatically prints the *device-name:/Are you sure?* message before it begins any operation. The device name that prints out in the message is the physical name of the device you specify in the command line. Therefore, if you use a logical device name in the command line, the device name that FORMAT displays in the verification message is different from the name you type. If you want the operation to continue, type a Y followed by a carriage return in response to the verification message. Any other response prevents the formatting operation from occurring and causes FORMAT to print its prompting asterisk again.

You can use FORMAT from an indirect command file. To satisfy the *device-name:/Are you sure?* message, enter a Y as the next line of the indirect file immediately following the FORMAT command line. You can suppress the verification message completely by using the /Y option in the FORMAT command line. If you use /Y, you do not need to enter the Y on the following line.

After you format a disk, you should use the INITIALIZE command to prepare the volume for use with RT–11. See Chapter 4 for more information on the INITIALIZE command.

18.2 Options

Options that you specify in a command line to the FORMAT program perform several functions. Table 18–1 summarizes these options and the operations they perform. You can combine these options, if necessary, in any order. More detailed explanations of the options are arranged alphabetically by option name in the sections that follow the table.

Table 18–1: FORMAT Options

Option	Section	Explanation
none	18.2.1	If you do not supply an option, FORMAT formats the volume you specify. If you specify an RX01 or RX02 diskette, the default operation that occurs is double-density diskette formatting. You can use /Y and /W with the default operation.
/P:n	18.2.2	Pattern verification option, where n represents an octal integer in the range 0 to 377. The option specifies the specific 16-bit word pattern that FORMAT uses to write to the volume, and read from the volume, during the process of verification. If you do not use this option, FORMAT defaults to /P:200.

(continued on next page)

Table 18–1: FORMAT Options (Cont.)

Option	Section	Explanation
/S	18.2.2	Single-density option. This option formats a diskette in a single-density format.
/V[:ONL]	18.2.3	Verification option. When you specify /V in the command line, FORMAT first formats the specified volume, then verifies it. If you specify /V:ONL, FORMAT only verifies the specified device. Note that you can use /V:ONL with any disk, diskette, DECtape, or DECtape II.
/W	18.2.4	Wait option. This option permits you to substitute another disk for the volume you specify in the command line, format the second volume, then replace the original volume.
/Y	18.2.5	No query option. This option suppresses the *Are you sure?* message FORMAT automatically prints before each operation.

18.2.1 Default Format

To format diskette in double-density mode, specify the device name in the command line. You can also use /Y to suppress the query message, and /W to pause for a volume substitution. The following example formats the diskette in DY: device unit 1 as a double-density diskette.

```
*DY1:
DY1:/FORMAT-Are you sure?Y
?FORMAT-I-Formatting complete
*
```

To format an RK05, RK06-7, or RP02-3 disk, specify the device name in the command line. You can also use /Y to suppress the query message and /W to pause for a volume substitution. The following example formats an RK05 disk in RK: device unit 1:

```
*RK1:
RK1:/FORMAT-Are you sure?Y
?FORMAT-I-Formatting complete
*
```

When you format an RK06 or RK07 disk, FORMAT lists the block numbers of all the bad blocks in the manufacturer's bad block table and in the software bad block table.

18.2.2 Pattern Verification Option (/P:n)

When you use the /P:n option with /V[:ONL] in the command line, you can specify the 16-bit word pattern you want FORMAT to use when it performs volume verification. The argument *n* represents an octal integer in the range 0 to 377 that specifies the pattern or successive patterns you want FORMAT to use. Table 18–2 lists the verification patterns FORMAT uses.

Table 18–2: Verification Bit Patterns

Pattern #	Bit Set	16-Bit Pattern
1	0	000000
2	1	111111
3	2	163126
4	3	125252
5	4	052525
6	5	007417
7	6	021042
8	7	104210

In /P:n, the number you specify for *n* indicates the value for the bit patterns to be run during verification. Bits set in /P:n select the patterns to be run. FORMAT converts the number you specify into a binary number; the number of each set bit specifies which patterns to run. For example, the number 25 translates to the binary number 010101. In the number 010 101, bits 0, 2, and 4 are set. As Table 18–2 shows, bit 0 specifies pattern 1; bit 2 specifies pattern 3; and bit 4 specifies pattern 5. If you specify /P:25, FORMAT runs patterns 1, 3, and 5. If you specify /P:255, FORMAT runs patterns 1, 3, 4, 6, and 8. If you specify /P:377, FORMAT runs all eight patterns during verification. If you do not use the /P:n option, FORMAT runs only pattern 8.

When you use the /P:n option, and you specify more than one pattern, FORMAT runs each pattern successively. After it completes verification, FORMAT prints at the terminal each bad block it found during each verification pass. The format of the verification report is:

```
PATTERN #x
----------
nnnnnn
```

In the example above, x represents the pattern number, and nnnnnn represents the bad block number. FORMAT makes a separate verification pass for each pattern it runs, and reports on each pass.

The sample command line that follows formats volume RK1: and verifies it with patterns 4, 5, and 6.

```
*RK1:/V/P:70
RK1:/FORMAT-Are you sure?Y
?FORMAT-I-Formatting complete
PATTERN #6
PATTERN #5
PATTERN #4
?FORMAT-I-Verification complete
*
```

The sample command line that follows verifies volume DL0: with pattern 2.

```
*DL0:/V:ONL/P:2
DL0:/VERIFY-Are you sure?Y
PATTERN #2
?FORMAT-I-Verification complete
```

18.2.3 Single Density Option (/S)

Use /S to format a diskette in single-density mode. You can also use /Y to suppress the query message and /W to pause for a volume substitution. The following example formats the diskette in DY: device unit 1 as a single-density diskette.

```
*DY1:/S
DY1:/FORMAT-Are you sure?Y
?FORMAT-I-Formatting complete
*
```

18.2.4 Verification Option (/V[:ONL])

Use the /V[:ONL] option to provide a verification of all blocks on a volume immediately following formatting. If you use the optional argument :ONL, FORMAT executes only the verification procedure. Although FORMAT can format only a limited assortment of storage volumes, it can verify any disk, diskette, DECtape, or DECtape II.

In the process of verifying a storage volume, FORMAT first writes a 16-bit word pattern on each block of the specified volume, and then reads each pattern. For each read or write error it encounters, FORMAT prints at the terminal the block number for each block that generated the error.

NOTE

FORMAT destroys data on any storage volume it verifies.

The following command line uses /V to verify an RK05 disk after formatting.

```
*RK0:/V
RX0:/FORMAT-Are you sure?Y
?FORMAT-I-Formatting complete
PATTERN #8
?FORMAT-I-Verification complete
```

The next example uses /V:ONL to verify, but not format, an RX01.

```
*DX1:/V:ONL
DX1:/VERIFY-Are you sure?Y
PATTERN #8
?FORMAT-I-Verification complete
```

18.2.5 Wait Option (/W)

Use /W to pause before formatting begins in order to substitute a second volume for the disk you specify in the command line. This is useful for single-disk systems. After the FORMAT program accepts your command line, it pauses while you exchange volumes. Type a Y followed by a carriage return in response to the *CONTINUE?* prompt when you are ready for formatting to begin. When formatting completes, the program pauses again while you

replace the original volume. Respond to the *CONTINUE?* prompt with a Y followed by a carriage return. You can combine /W with any other option. The following example formats the diskette in DY: device unit 1 as a single-density diskette.

```
*DY1:/W/S
DY1:/FORMAT-Are you sure?Y
Insert volume you wish to format.    CONTINUE(Y/N)?Y
?FORMAT-I-Formatting complete
Replace original volume.             CONTINUE(Y/N)?Y
*
```

18.2.6 Noquery Option (/Y)

Use /Y to suppress the *Are you sure?* query message FORMAT prints before each operation begins. When you use /Y, formatting begins as soon as FORMAT accepts and interprets your command line. The following example formats the diskette in DY: device unit 1 as a double-density diskette.

```
*DY1:/Y
?FORMAT-I-Formatting complete
*
```

Part V
System Jobs

Part V describes two utilities that you can run as system jobs: the Error Logger and the Queue Package. System job support is a special feature, available only through the system generation process. Because system jobs run in the foreground, they must run under an FB or XM monitor. For an in-depth description of the system job feature, see the *RT–11 Software Support Manual.*

Chapter 19 describes the Error Logger, which keeps statistical records of all I/O transfers, I/O errors, memory parity errors, and cache memory errors. Chapter 20 describes the Queue Package, which sends files for output to any valid RT–11 device.

Note that you can run these utilities as foreground jobs, but running them as system jobs enables you to run a foreground and a background job in addition.

Chapter 19
Error Logging

The Error Logger utility monitors the hardware reliability of the system. The Error Logger keeps a statistical record of all I/O operations on devices that it supports. In addition to keeping these statistics, the Error Logger detects and records memory parity or cache errors and any errors that occur during I/O operations. At intervals you determine, the Error Logger produces individual and/or summary reports on some or all of these errors. The Error Logger is available only as a special feature; that is, you must perform the system generation process to enable Error Logging. It runs only with the foreground/background or extended memory monitors, as either a foreground or system job.

19.1 Uses

Error logging reports are useful for maintaining the hardware on which RT–11 runs. Problems such as line noise, static discharges, or inherently error-prone media can cause recoverable errors on systems that are otherwise functioning normally. By studying error logging reports, you can learn to distinguish these errors from those that might be symptoms of an impending device failure. Also, some recoverable errors that are insignificant in themselves might be related to other more serious errors; their effects might not be immediately apparent to you. Information contained in the reports about each error and about the status of the system when the error occurred may alert you to a previously unforeseen hardware problem.

Sometimes a device fails so quickly that the Error Logger is unable to predict the failure in time for you to prevent it. In this case, you can determine the cause more quickly if a report is available that describes the errors that occurred immediately prior to the failure.

In general, the Error Logger:

- Detects each I/O transfer and I/O error as it occurs
- Gathers information about each I/O transfer or error
- Stores the information in a file
- Formats the information to produce a report

NOTE

Because the Error Logger records data on each I/O transfer, thereby using additional computer time and memory, you may wish to use the Error Logger only when you experience difficulty with a device. Keeping a backup system volume on which the Error Logger is enabled makes this easy.

19.2 Error Logging Subsystem

The Error Logger consists of three programs and a statistics file. When you run the Error Logger, you coordinate these programs to gather I/O and error-related information into its statistics file and create the error report you want. The Error Logger names the statistics file it creates ERRLOG.DAT. At any time you specify, you can direct the Error Logger to create error reports from the information it has gathered in ERRLOG.DAT. The names and functions of the three Error Logger programs follow.

EL — A foreground or system job that gathers information about every I/O transfer and system error and stores it in ERRLOG.DAT. The EL job communicates with a device handler as each I/O transfer occurs, and it gathers from the handlers all the necessary statistics for a complete error report. When you initiate error logging, EL instructs you to start up the second Error Logging program, ELINIT.

ELINIT — A background job that creates and maintains the statistics file, ERRLOG.DAT. You can direct ELINIT to initialize ERRLOG.DAT every time you have a session at the terminal, or you can direct ELINIT to continue compiling statistics in ERRLOG.DAT on a daily basis.

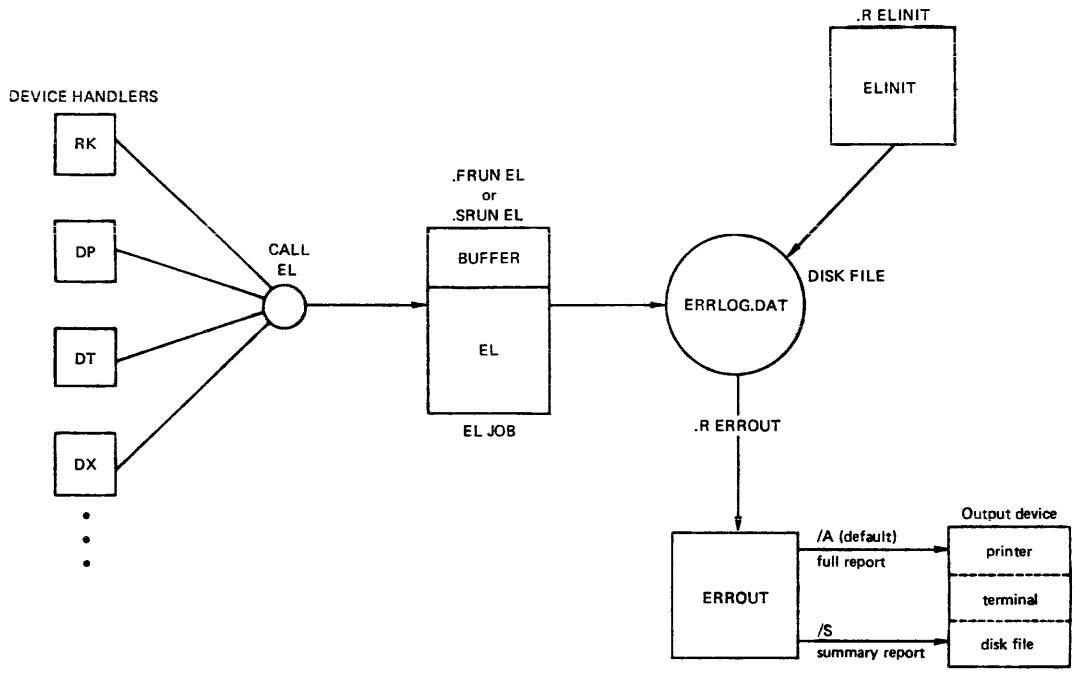
When you run ELINIT, it prompts you for the information it needs to maintain ERRLOG.DAT's size. By default, ELINIT allocates 100 decimal blocks for ERRLOG.DAT. Each time you run ELINIT, it prints a message that tells how many of those 100 blocks are filled. If ERRLOG.DAT fills to its limit, the EL job is unable to store more information in it. So that you can increase ERRLOG.DAT's size, ELINIT prompts you for a file size change each time you run the program.

If you rearrange your RT–11 configuration, or if you do something to alter the device handlers, ELINIT may print a message indicating that it must initialize ERRLOG.DAT to make the statistics it has been maintaining compatible with the new configuration. When this happens, ELINIT renames the ERRLOG.DAT it formerly maintained to ERRLOG.TMP and creates a new ERRLOG.DAT. The Error Logger can still create a report from ERRLOG.TMP.

ERROUT — A background job that creates a report from the statistics in ERRLOG.DAT or any file of that format. When you run ERROUT, you can direct the program to list the error report at the terminal or to create a file for the error report. You can also indicate whether you want a detailed report on each error that occurred or simply a summary report.

Figure 19–1 provides a diagram of the Error Logging subsystem.

Figure 19–1: Error Logging Subsystem



19.3 Calling and Using the Error Logger

The Error Logger runs only with the FB or XM monitors as a foreground or system job. To run the Error Logger as a foreground job, call the Error Logger from the system device by typing in response to the keyboard monitor dot (.):

```
FRUN EL <RET>
```

To run the Error Logger as a system job, type in response to the keyboard monitor dot:

```
SRUN EL <RET>
```

The Error Logger returns with a prompt, telling you how to initiate the error logging process.

```
To initiate Error Logging, RUN ELINIT
```

To halt the Error Logger if it is running as a foreground job, type a CTRL/F followed by two CTRL/Cs. If it is running as a system job, you can type a CTRL/X and then specify EL as the system job you want to halt (followed by two CTRL/Cs).

NOTE

If you have not installed the system on which you want to run the error logger, check EL's file type on a directory listing of your system volume. In order to run EL as a system

job, RT–11 assumes EL has a .SYS file type. If the EL program on your system volume has a .REL file type, and you want to run EL as a system job, either:

1. rename EL.REL to EL.SYS and type SRUN EL, or
2. type SRUN EL.REL.

Likewise, if you want to run EL as a foreground job, and the EL program on your system volume has a .SYS file type, either:

1. rename EL.SYS to EL.REL and type FRUN EL, or
2. type FRUN EL.SYS.

19.3.1 Using ELINIT

After you type RUN ELINIT (or R ELINIT) followed by a carriage return, ELINIT returns with a prompt. This prompt asks you to specify which device you want the statistics file ERRLOG.DAT written to. The format of this prompt follows.

```
What is the name of the device for the ERRLOG.DAT file <SY>?
```

Type a carriage return in response to the last prompt if you want ELINIT to write ERRLOG.DAT to the system device.

ELINIT then prints a message indicating how many blocks allocated to ERRLOG.DAT are in use. This message is followed by a prompt asking you if you want ELINIT to initialize ERRLOG.DAT. The format of the block usage message and the initialization prompt follows (where *xx* represents the number of blocks in use).

```
xx blocks currently in use of xx possible total in ERRLOG.DAT file

Do you want to zero the ERRLOG.DAT file and re-initialize (YES/NO) <NO>?
```

Type YES followed by a carriage return if you want ELINIT to initialize ERRLOG.DAT. When ELINIT initializes ERRLOG.DAT, it does not create a backup file for the statistics that were present prior to initialization. Enter a carriage return or type NO followed by a carriage return if you want ELINIT to retain the statistics already compiled in ERRLOG.DAT.

ELINIT proceeds by issuing the following prompt, asking you to indicate the number of blocks you want ELINIT to allocate to ERRLOG.DAT:

```
How many blocks for the ERRLOG.DAT file <100>?
```

Type a carriage return if you want ERRLOG.DAT's file size to remain at the default 100 blocks. If you want the file to be a different size, you can specify the number of blocks you want the file to have, followed by a carriage

return. The only size limitation for ERRLOG.DAT is the amount of available space on the device in which it resides.

NOTE

Because of a rearrangement of your RT–11 configuration or bad header information in ERRLOG.DAT, it may be necessary for ELINIT to initialize ERRLOG.DAT even if you do not want it to. In this case, ELINIT automatically renames the current ERRLOG.DAT to ERRLOG.TMP, prints a message indicating it has done so, and returns the prompt *How many blocks for the ERRLOG.DAT file <100>?*.

After you have responded to the file size prompt, ELINIT prints the following message:

```
RT-11 V04.0 ERROR LOGGING INITIATED
```

After the Error Logger has printed the last message, you can proceed.

19.3.2 Using ERROUT

The Error Logger program, ERROUT, creates a report from the information compiled in the file ERRLOG.DAT. To call ERROUT from the system device, type in response to the keyboard monitor dot (.):

```
RUN ERROUT <RET>
```

The Command String Interpreter prints an asterisk at the left margin of the terminal and waits for you to enter a command string according to the following general syntax:

*[output-filespec =][input-filespec]/option

where:

output-filespec — represents the device to which you want ERROUT to type the report. If you do not specify an output device, ERROUT prints the report at the terminal. If you specify a file name, ERROUT writes the error report to that file

input-filespec — represents ERRLOG.DAT or any file of the Error Logger statistics file format. (Thus, you can rename ERRLOG.DAT at any time and save it for later report formatting.) If you do not specify an input file, ERROUT assumes ERRLOG.DAT

option — is one of the options listed below.

- /A — creates a report on each error in addition to a summary report of the errors and I/O transfers that occurred with each device.

/F:date — use to create an error report for errors logged from the date you specify. Specify the date in the form *dd:mmm:yy*, where *dd* represents the two-digit day, *mmm* represents the first three letters of the month, and *yy* represents the last two digits of the year. ERROUT interprets the date you enter as octal; use a decimal point with the day and year to indicate the date is in decimal. If you do not use /F:date, ERROUT creates a report starting with the first error logged in the work file.

/S — creates only a summary report of the errors and I/O transfers that occurred with each device.

/T:date — use to create an error report for errors logged up to the date you specify. Specify the date as with the /F:date option above. If you do not use /T:date, ERROUT creates a report that includes the last error logged in the work file.

If you enter only a carriage return in response to the CSI asterisk, ERROUT types a full report from ERRLOG.DAT at the terminal.

19.4 Report Analysis

This section provides a line-by-line analysis of each different report the Error Logger creates. Basically, there are three report categories:

- Storage Device Error Report
- Memory Error Report
- Summary Report

19.4.1 Storage Device Error Report

When a device handler encounters an error during an I/O transfer, it automatically retries that transfer as many as eight times (the actual number of times a device retries an unsuccessful transfer depends on the particular device handler). The Error Logger creates a report for each unsuccessful attempt.

Figure 19–2 provides an example of a storage device error report. This example is a report of the second attempt for a read operation on an RX02 double-density diskette. Table 19–1 tells what some of the lines in the report

mean. For ease of reference, each line in this example report is numbered (although lines in the actual report are not numbered).

Figure 19–2: Sample Storage Device Error Report

```
1  ****************************************************************
2  DISK DEVICE ERROR
3  LOGGED 27-APR-79 00:06:20
4  ****************************************************************
5
6  UNIT IDENTIFICATION
7          PHYSICAL UNIT NUMBER                              000001
8          TYPE                                          RX211/RX02
9  SOFTWARE STATUS INFORMATION:
10         RETRY 6. OF 8. POSSIBLE TOTAL
11
12 DEVICE INFORMATION
13         REGISTERS:
14         RX2CS              114560
15         RX2DB              010400
16         RX2ES              000400
17
18         ACTIVE FUNCTION                    READ
19         BLOCK                            000001
20         PHYSICAL BUFFER ADDRESS START    003514
21         TRANSFER SIZE IN BYTES              512.
```

Table 19–1 explains each line in the sample report shown in Figure 19–2.

Table 19–1: Line-by-Line Analysis of the Sample Storage Device Error Report

Line	Explanation
1–4	Report header. Includes the date and time error was logged.
6–8	Unit identification. Identifies the drive number, the device controller, and the storage device type.
9–10	Retry count. Tells which retry of the total possible for the device the device handler has attempted for the I/O transfer. There is a report for each retry the device handler attempts. Notice that the retry count is listed in descending order; that is, the first retry for this example is listed as retry 7. The second retry is listed as retry 6, and so on.
12	Labels the section on device information. The lines that follow provide statistics on the device registers and address information.
13–16	Register contents. Each device has a number of hardware registers, the contents of which are listed in these lines.
18	I/O transfer type. Tells whether the I/O transfer was a read or write operation.
19	Device block number. Tells which device block the error occurred in.
20	Physical buffer start address. Tells the physical address in memory of the user data buffer for this I/O transfer.
21	Transfer size in bytes. Tells the size in bytes of the unit of data the device handler has attempted to transfer.

19.4.2 Memory Error Report

There are two kinds of memory errors for which the Error Logger creates reports: memory parity errors and cache memory errors. Figure 19–3 provides an example of a memory parity error report. As with the storage device report, this listing is numbered in the manual to aid in describing its contents. The listings that you obtain do not have line numbers.

Figure 19–3: Sample Memory Parity Error Report

```
1  ************************************************************************
2  MEMORY PARITY ERROR
3  LOGGED 27-APR-79 00:06:20
4  ************************************************************************
5
6  DEVICE INFORMATION
7           MEMORY REGISTERS:
8           ADDRESS       CONTENTS
9           172100        100001
10
11          ERROR TYPE IS MEMORY
```

Table 19–2 tells what each line in the last report shown in Figure 19–3 means.

Table 19–2: Line-by-Line Analysis of the Sample Memory Error Report

Line	Explanation
1–4	Report header. Tells the date and time the error was logged.
7–9	Memory parity register contents. Identifies the memory parity control and status register(s) involved in this error and gives their contents.
11	Error type. Tells whether the error was a memory error or a cache memory error (see the following cache memory report for cache memory statistics).

The report in Figure 19–4 is an example of the report the Error Logger creates when it logs a cache memory error.

Figure 19–4: Sample Cache Memory Error Report

```
1  ************************************************************************
2  CACHE MEMORY ERROR
3  LOGGED 27-APR-79 00:06:20
4  ************************************************************************
5
6  DEVICE INFORMATION
7           MEMORY REGISTERS:
8           ADDRESS       CONTENTS
9           172100        100001
10
11          MEMORY SYSTEM ERROR REGISTER:     000200
12          CACHE CONTROL REGISTER:           000000
13          HIT/MISS REGISTER:                000032
14
15          ERROR TYPE IS CACHE
```

Lines 11 through 13 in Figure 19–4 tell the contents of the cache memory registers (see the *PDP–11 Processor Handbook* for more information on the cache memory registers). Line 15 indicates that the memory error was in cache memory.

19.4.3 Summary Error Report

The summary error report provides statistics for all the devices the Error Logger supports. These statistics include counts for successful and unsuccessful I/O transfers for storage devices, and error counts for memory errors. The report consists of three sections:

- device statistics
- memory statistics
- report file environment and error count

Figure 19–5 provides an example of a summary error report.

Figure 19–5: Sample Summary Error Report for Device Statistics

```
1  ********************************************************************************
2  DEVICE STATISTICS
3  LOGGED SINCE 27-APR-79 00:05:42
4  ********************************************************************************
5
6  UNIT IDENTIFICATION
7          PHYSICAL UNIT NUMBER                                          000000
8          TYPE                                                        RK11/RK05
9
10 DEVICE STATISTICS FOR THIS UNIT:
11         NUMBER OF ERRORS LOGGED                  0.
12         NUMBER OF ERROR RECEIVED                 0.
13         NUMBER OF READ SUCCESSES                49.
14         NUMBER OF WRITE SUCCESSES                0.
15
16 UNIT IDENTIFICATION
17         PHYSICAL UNIT NUMBER                                          000000
18         TYPE                                                      RX211/RX02
19
20 DEVICE STATISTICS FOR THIS UNIT:
21         NUMBER OF ERRORS LOGGED                  1.
22         NUMBER OF ERRORS RECEIVED                1.
23         NUMBER OF READ SUCCESSES                 2.
24         NUMBER OF WRITE SUCCESSES                0.
25
26 UNIT IDENTIFICATION
27         PHYSICAL UNIT NUMBER                                          000001
28         TYPE                                                      RX211/RX02
29
30 DEVICE STATISTICS FOR THIS UNIT:
31         NUMBER OF ERRORS LOGGED                 16.
32         NUMBER OF ERRORS RECEIVED               16.
33         NUMBER OF READ SUCCESSES                 0.
34         NUMBER OF WRITE SUCCESSES                0.
```

Notice that the Error Logger provides summary statistics for each device. Notice for each device, the count of the number of errors logged and the count of the number of errors received can be different. Sometimes, the

Error Logger may receive an error but be unable to log it. This is usually due to full buffers or some other momentary software limitation. However, even if the Error Logger is unable to log an error, it is at least able to inform you of this fact.

Figure 19–6 provides an example of the second section of the summary error report, memory statistics. This report immediately follows the report on device statistics.

Figure 19–6: Sample Summary Error Report for Memory Statistics

```
1  ********************************************************************
2  MEMORY STATISTICS
3  LOGGED SINCE 27-APR-79 00:05:42
4  ********************************************************************
5
6  STATISTICS:
7           NUMBER OF MEMORY PARITY ERRORS                    5.
8           NUMBER OF CACHE ERRORS                            0.
```

Figure 19–7 provides an example of the third section of the error report summary, the report file environment and error count.

Figure 19–7: Sample Report File Environment and Error Count Report

```
1  REPORT FILE ENVIRONMENT:
2           INPUT FILE                    SY0:ERRLOG.DAT
3           OUTPUT FILE                   RK0:ERRORS.TXT
4           OPTIONS                              /A
5           DATE INITIALIZED                27-APR-79
6           DATE OF LAST ENTRY              27-APR-79
7  TOTAL ERRORS LOGGED                              22.
8  MISSED ENTRIES                                    0.
9  MISSED ERROR ENTRIES                              0.
10 UNKNOWN DEVICE STATISTICS ENTRIES                 0.
11 UNKNOWN ERROR RECORD ENTRIES                      0.
```

The segment, shown in Figure 19–7 report file environment, provides information concerning the input report file name (usually ERRLOG.DAT or ERRLOG.TMP) and the output report file name (any name that you specify in the initial ERROUT command line). In line 5, the report tells when the input report file was initialized, and in line 6, the date of the last error entry to the input report file.

Lines 7 through 11 count additional error count statistics. Line 8 counts the number of missed entries.

Line 9 counts the number of missed error entries.

Line 10 provides a count of unknown device statistics entries. An unknown device statistics entry occurs when ERROUT does not recognize the device identifier byte the EL program recorded in the statistics portion of the ERRLOG.DAT file. (All DIGITAL-distributed device handlers that support Error Logging can be identified by ERROUT, so this problem occurs most often with user-written handlers. See the *RT–11 Software Support Manual* for details on adding a device to ERROUT.)

Line 11 keeps a count of the unknown error record entries. This condition occurs when the ERROUT task cannot identify a device error recorded in the ERRLOG.DAT file. (Again, this condition occurs most often with user-written handlers.)

Chapter 20
Queue Package

The Queue Package is a utility you can use for sending files to any valid RT–11 device. Although the Queue Package is particularly useful for queuing files for printing, queuing is not restricted to a line printer or any other serial device.

The Queue Package consists of two programs and a work file that contains the lineup of files, or queue, waiting to be output:

QUEUE	Queues and sends the files you specify. QUEUE runs as a foreground or system job.
QUEMAN	A background job that processes command lines and file specifications you enter, and sends that information to QUEUE. It serves as the interface between you and the Queue Package.
QUFILE.TMP	Contains the queue for the files QUEUE sends to the device(s) you specify.

The Queue Package runs only with the FB or XM monitor.

NOTE

To prevent QUEUE and another job from intermixing output on the same nonfile structured device, use the LOAD command to assign exclusive ownership of a device to QUEUE.

20.1 Calling and Using the Queue Package

To use the Queue Package, you must first call and run QUEUE from the system volume as either a foreground or system job. (Note that system job support is a special feature. You can perform the system generation process to build a monitor and handlers that support system jobs.) You can then call and run QUEMAN when you are ready to output files.

20.1.1 Calling and Running QUEUE

To run QUEUE as a foreground job, call QUEUE from the system volume by typing in response to the keyboard monitor dot (.):

```
FRUN QUEUE <RET>
```

To run QUEUE as a system job, type in response to the keyboard monitor dot:

```
SRUN QUEUE <RET>
```

To halt QUEUE, see Section 20.2.1.

NOTE

If you yourself did not install the system on which you want to run the Queue Package, check QUEUE's file type on a directory listing of your system volume. In order to run QUEUE as a system job, RT–11 assumes QUEUE has a .SYS file type. If the QUEUE program on your system volume has the .REL file type, and you want to run QUEUE as a system job, either:

1. rename QUEUE.REL to QUEUE.SYS and type SRUN QUEUE, or
2. type SRUN QUEUE.REL.

Likewise, if you want to run QUEUE as a foreground job, and QUEUE has a .SYS file type, either:

1. rename QUEUE to QUEUE.REL and type FRUN QUEUE, or
2. type FRUN QUEUE.SYS.

20.1.2 Calling and Running QUEMAN

To call QUEMAN from the system volume, type in response to the keyboard monitor dot (.):

```
R QUEMAN <RET>
```

The Command String Interpreter prints an asterisk at the left margin of the terminal indicating it is ready to accept input. Enter a command string according to this general syntax:

[dev:[jobname[/options]]=][filespec][/options][,filespec[/options]...]

where:

dev: represents any valid RT–11 device (The default output device is LP0:.)

jobname represents the output job name. This is the logical name for all the files specified in the command. If you send a job to a file-structured device, QUEUE uses this name as the file name of the job, and assigns a .JOB file type. If you do not specify a job name, QUEMAN uses the file name of the first input file. The job name can have up to six characters

filespec represents the input file specification. If you do not specify a file type, QUEMAN assumes a .LST file type

options represents one or more of the options from Table 20–1

If you use commas in place of file specifications, QUEMAN ignores all remaining file specifications on the command line. (Note, however, that if your command string consists of several lines, entering commas in place of a file specification does not affect file specifications on subsequent lines in the command string. Using commas in place of a file specification affects only those remaining files in that particular command line.)

20.2 QUEMAN Options

Table 20–1 summarizes the options you can use in the QUEMAN command line. The sections that follow Table 20–1 provide detailed explanations and examples of each option. Note that some of the options are position-dependent — that is, their function depends on where you place them in the command line.

Table 20–1: QUEMAN Options

Option	Section	Function
/A	20.2.1	Halts QUEUE.
/D	20.2.2	Deletes the input file(s) after printing. This option is position-dependent.
/H:n	20.2.3	Prints *n* banner pages for each specified input file, where *n* is a decimal number. This option is position-dependent.
/K:n	20.2.4	Prints *n* copies of each specified file, where *n* is a decimal number. This option is position-dependent.
/M	20.2.5	Removes a job from the queue.
/L	20.2.6	Lists the contents of the queue.
/N	20.2.7	Specifies no banner pages for the input file(s).
/P	20.2.8	Sets two Queue Package default values: the number of banner pages, and whether you want QUFILE.TMP deleted when you halt QUEUE.
/R	20.2.10	Resumes sending the current job after it has been suspended, or restarts the current file in the job being sent.
/S	20.2.9	Suspends output at the end of the current file.
//	20.2.11	Continues command on the next line.

If QUEUE is sending a job that has multiple input files to a file-structured volume, QUEUE concatenates all the input files into one output file having the name of the job.

20.2.1 Halting QUEUE (/A)

When you type /A in response to the CSI asterisk, QUEMAN halts QUEUE. If you use /A while a job is printing, the Queue Package halts output. If QUEUE is running as a foreground job, using /A has the same effect as typing CTRL/F and two CTRL/Cs. If QUEUE is running as a system job, using /A has the same effect as typing CTRL/X and then specifying QUEUE as the system job you want to direct input, followed by two CTRL/Cs.

The following example halts QUEUE

```
.R QUEMAN
*/A <RET>
```

If you use CTRL/Cs to halt QUEUE, this may take a few seconds because QUEUE halts differently from other foreground or system jobs.

20.2.2 Deleting Input Files After Printing (/D)

Use the /D option to delete input files after QUEUE has sent them. This option is position-dependent. If you use it with the job name, /D applies to all the input files. If you use it with an input file specification, /D applies only to that input file.

The following example deletes all input files after they have been sent:

```
*MYJOB/D=FILE1,FILE2,FILE3 <RET>
```

The following example deletes FILE1 and FILE3 but retains FILE2 after QUEUE has sent them:

```
*MYJOB=FILE1/D,FILE2,FILE3/D <RET>
```

20.2.3 Printing Banner Pages (/H:n)

Use the /H:n option to print banner pages for the input files you specify, where *n* is a decimal number selecting the number of banner pages. This option is position-dependent. If you use /H:n with the job name, QUEUE prints banner pages for each input file. If you use /H:n with an input file specification, QUEUE prints *n* banner pages for that file, and prints the default number of banner pages for the remaining input files. (Note that you set the default number of banner pages with the /P option, described in Section 20.2.8.)

The sample command line that follows prints four banner pages for each input file:

```
*LAUGHN/H:4=ROWAN.TXT,MARTIN.TXT<RET>
```

The following sample command prints four banner pages for MARTIN.TXT and one for ROWAN.TXT:

```
*LAUGHN=ROWAN.TXT,MARTIN.TXT/H:4<RET>
```

Note that QUEUE never prints a banner for the job; it prints banners only for the input files.

20.2.4 Printing Multiple Copies (/K:n)

Use the /K:n option to specify the number of copies of the input files you specify, where *n* is a decimal number. The /K:n option is position-dependent. If you use /K:n with the job name, QUEUE prints *n* copies of each input file. If you use /K:n with an input file specification, QUEUE prints *n* copies of that particular file.

The next command line prints four copies of LAUREL.LST and four copies of HARDY.LST:

```
*JOB/K:4=LAUREL,HARDY<RET>
```

The following sample command line prints four copies of LAUREL.LST and the default number of copies of HARDY.LST:

```
*JOB=HARDY,LAUREL/K:4 <RET>
```

20.2.5 Removing a Job from the Queue (/M)

Use the /M option to remove a job from the queue. When you use this option, specify the job name followed by /M and the equal sign (=). The following example removes the job LAB4 from the queue:

```
*LAB4/M= <RET>
```

When you use /M, you do not have to specify the input files, only the job name. You remove all the files associated with the job name.

20.2.6 Listing the Contents of the Queue (/L)

Use the /L option to get a listing of the contents of the queue. The listing gives the output device, job name, input files, job status, and number of copies for each job that is in the queue. The job STATUS column prints *P* if the job is currently being sent, *S* if the job being sent is suspended, or *Q* if the job is waiting to be sent. If you have a large queue and your console is a video terminal, you can use the keyboard CTRL/S and CTRL/Q commands to control the scrolling of the listing.

The sample command line that follows lists the queue:

```
*/L <RET>
DEVICE    JOB     STATUS    COPIES  FILES

 LP0:     LAB2      P          1     PASS3 .LST
                               2     PASS4 .LST
                               2     PASS5 .LST
 LP0:     HODG      Q          3     MESMAN.DOC
```

(continued on next page)

```
MT1:    JUDITH    Q        2     PART1 .DOC
                           2     PART2 .DOC
DT1:    SZYM      Q        1     REFMAN.DOC
LP0:    JOYCE     Q        1     SSM   .DOC
                                 DOCPLN.DOC
```

20.2.7 No Banner Pages Option (/N)

Use the /N option to specify that you do not want QUEUE to print any banner pages for the input file(s). Use /N if you have previously set the default number of banner pages with the /P option (see Section 20.2.8). The /N option is position-dependent; that is, if you use it after the job name, it applies to each input file. Use /N after an input file to apply to only the particular file.

The following example uses /N to specify no banner pages for each file in the job, MYJOB2.

```
*MYJOB2/N=PASS1,PASS2,PASS3
```

The /N option has the same effect as /H:0 (see Section 20.2.3).

20.2.8 Setting Queue Package Defaults (/P)

Use the /P option to set defaults for two values:

1. Number of banner pages printed for each input file
2. Whether you want the work file, QUFILE.TMP, deleted when you halt QUEUE (Note that QUFILE.TMP contains the lineup of files, or queue, waiting to be sent to an output device.)

When you type /P in response to the CSI asterisk, QUEMAN prints the following prompt at the terminal:

```
1) Number of banner pages ?
```

QUEUE uses the number you type as the default number of banner pages it prints for each file it sends to a device. If you type only a carriage return, QUEMAN assumes 0. This value remains in effect until the work file, QUFILE.TMP, is deleted (see below).

After you have responded to the previous prompt, QUEMAN prints the following prompt at the terminal:

```
2) Delete workfile ?
```

If you type N followed by a carriage return, or only a carriage return, QUEUE maintains the current QUFILE.TMP after you halt QUEUE. That is, if you start QUEUE later, QUFILE.TMP retains the queue it had prior to the halt. By maintaining QUFILE.TMP between the times QUEUE is halted, you have an automatic queue restart capability. This value remains in effect until you reset it.

If you type Y followed by a carriage return, QUEUE deletes QUFILE.TMP when you halt QUEUE. The next time you start QUEUE, it creates a new QUFILE.TMP. This value remains in effect only until the next time you start QUEUE.

20.2.9 Suspending Output (/S)

Use the /S option to suspend output of a job being sent. When you type /S in response to the CSI asterisk, QUEUE suspends output only after it has completed output of the current file in the job. This option is useful if you want access to an output device while a large job is being sent to it.

If you suspend a job that is being sent to a storage volume, QUEUE creates a file of the job output thus far, making it a permanent file. It uses the job name as the file name and assigns a .JOB file type. (If you do not specify a job name, QUEUE uses the name of the first input file.)

To resume output, use the /R option (Section 20.2.10).

20.2.10 Resuming/Restarting Output (/R)

Use the /R option either to resume output of a suspended job, or to restart output of the current file in the job from the beginning of the file. Note that a job resumes if you previously suspended it with /S, and a job restarts if you have not previously suspended it.

The effect of the /R option depends on the particular output device involved. If you are resuming output of a job to a line printer, QUEUE resumes output with the subsequent file in the job. If you are restarting output of a job to a line printer, QUEUE goes back to the beginning of the file currently being sent and prints the file again. If more than one copy of a particular file is being sent, the number of copies specified is printed again when you restart output, regardless of how many copies were sent prior to restarting.

NOTE

If you restart or resume a job being sent to an RT–11, or file-structured volume, and there are multiple input files in the job, you will lose files that were output prior to the restart.

Resuming a job with multiple input files when the job is being sent to an RT–11, file-structured volume can be useful if the volume involved is too small to contain the entire job. You can suspend the job being sent (using the /S option), change volumes, and resume output of the remainder of the job on the new volume. QUEUE uses the same file name for both parts of the job.

20.2.11 Continuing a Command String (//)

Use the // option to continue a command string on subsequent lines. This option is useful if you want to output more files than you can specify on one line. When you want to include several lines in a CSI command string, type // at the end of the first line, and again at the end of the command string.

The following command string uses the // option:

```
*JOBNAM=LML1.MAC,LML2A.MAC// <RET>
*LML51.MAC,LML95.MAC <RET>
*LML4.MAC,LML56.MAC// <RET>
```

Part VI
Debugging and Altering Programs

Part VI of this manual consists of the following five chapters: On-Line Debugging Technique (ODT), Save Image Patch Program (SIPP), Object Module Patch Utility (PAT), Source Language Patch Program (SLP), and Patch Utility (PATCH). The five programs that these chapters describe can help you debug programs, examine or change assembled programs, and patch source programs.

Chapter 21 describes ODT. This program aids you in debugging assembly language programs. With ODT, you can control your program's execution, examine locations in memory and alter their contents, and search the object program for specific words.

Chapter 22 describes SIPP. You can use SIPP to examine or modify individual locations within programs linked with the RT–11 V04 linker. Using SIPP, you can also create an indirect command file that contains a patch and the commands necessary to install it.

Chapter 23 describes PAT, which patches or updates code in a relocatable binary object module. PAT accepts a file containing corrections or additional instructions and applies these corrections and additions to the original object module.

Chapter 24 describes SLP. SLP provides an easy way to make changes to source files. This program can accept an indirect command file created by the DIFFERENCES/SLP/OUTPUT:filespec command (or the SRCCOM /P option) to make two source programs match.

Chapter 25 describes PATCH. Patch can make code modifications to any RT–11 file. You use PATCH to examine and then change words or bytes in a file. PATCH's checksum feature is particularly useful when you are making a correction or improvement to an already existing executable program; it verifies that the changes you make are correct.

Chapter 21
On-Line Debugging Technique (ODT)

On-line Debugging Technique (ODT) is a program that aids in debugging assembly language programs. ODT performs the following tasks.

- Prints the contents of any location for examination or alteration.
- Runs all or any portion of an object program using the breakpoint feature.
- Searches the object program for specific bit patterns.
- Searches the object program for words that reference a specific word.
- Calculates offsets for relative addresses.
- Fills a single word, block of words, byte or block of bytes with a designated value.

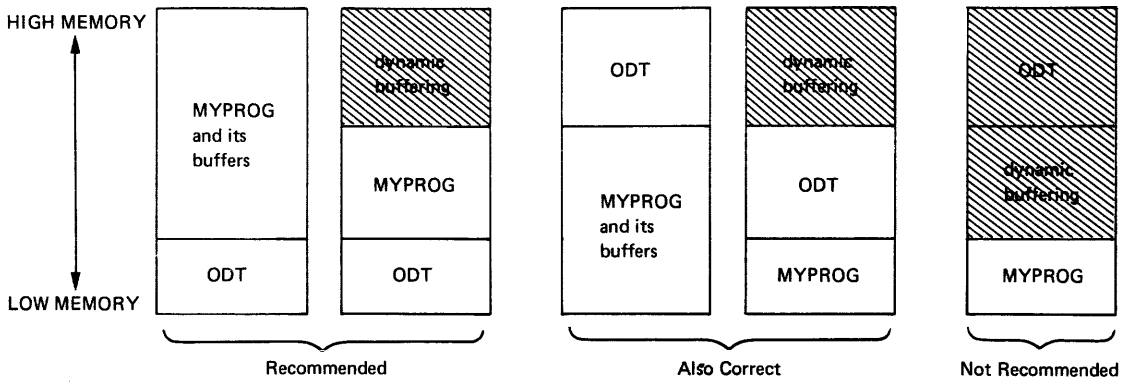
Make sure you have an assembly listing and a link map available for the program you want to debug with ODT. You can make minor corrections to the program on line during the debugging session, and you can then execute the program under the control of ODT to verify the corrections. If you need to make major changes, such as adding a missing subroutine, note them on the assembly listing and incorporate them in a new assembly.

See the *RT–11 Software Support Manual* for debugging the following routines and jobs: interrupt service routines, device handlers, multi-terminal jobs, extended memory and virtual jobs.

21.1 Calling and Using ODT

ODT is supplied as a relocatable object module. You can link ODT with your program (using the RT–11 linker) for an absolute area in memory and load it with your program. When you link ODT with your program, it is a good idea to link ODT low in memory relative to the program. If you do link ODT high in memory, be sure that the buffer space for your program is contained within program bounds. Otherwise, if your program uses dynamic buffering, program execution may destroy ODT in memory. Figure 21–1 shows the relationship between ODT and the program MYPROG in memory.

Figure 21–1: Linking ODT with a Program



Once loaded in memory with your program, ODT has three legal start or restart addresses. Use the lowest (O.ODT) for normal entry, retaining the current breakpoints. The next (O.ODT+2) is a restart address that clears all breakpoints and re-initializes ODT, thus saving the general registers and clearing the relocation registers. Use the last address (O.ODT+4) to reenter ODT. A reenter saves the processor status and general registers, and removes the breakpoint instructions from your program. ODT prints the bad entry (BE) error message. Breakpoints that were set are reset by the next ;G command. (;P is illegal after a BE message.) The ;G and ;P commands control program execution and are explained in Section 21.3.7.

The system uses as an absolute address the address of the entry point O.ODT shown in the linker load map.

NOTE

If you link ODT with an overlay-structured file, it should reside in the root segment so that it will always be in memory. Remove all breakpoints from the current overlay segment before execution proceeds to another overlay segment. A breakpoint inserted in an overlay is destroyed if it is overlaid during program execution.

The following example links ODT low in memory relative to MYPROG, creating the executable module MYPROG.SAV. Running MYPROG causes ODT to start automatically.

```
.LINK/MAP:TT:/DEBUG MYPROG
RT-11 LINK  V06.01      Load Map
MYPROG.SAV      Title:  ODT     Ident:  V04.00

Section  Addr   Size    Global  Value   Global  Value   Global  Value

. ABS.   000000 001000   (RW,I,GBL,ABS,OVR)
$ODT$    001000 006152   (RW,I,LCL,REL,CON)
                         O.ODT   001232
PROG     007152 002052   (RW,I,LCL,REL,CON)
                         START   007152
```

```
Transfer address = 001232, High limit = 011222 = 2377. words

.

.R MYPROG

 ODT  V04.00
*
```

The following example links MYPROG low in memory relative to ODT and specifies O.ODT as the transfer address. Running MYPROG causes ODT to start automatically. The advantage to this method is that MYPROG is loaded at its normal, execution-time address.

```
.LINK/MAP:TT: MYPROG,ODT/TRANSFER
Transfer symbol? O.ODT
RT-11 LINK  V06.01      Load Map
MYPROG.SAV      Title:  MYPROG  Ident:  V04.00

Section  Addr   Size    Global  Value   Global  Value   Global  Value

. ABS.   000000 001000   (RW,I,GBL,ABS,OVR)
PROG     001000 002052   (RW,I,LCL,REL,CON)
                         START   001000
$ODT$    003052 006152   (RW,I,LCL,REL,CON)
                         O.ODT   003304

Transfer address = 003304, High limit = 011222 = 2377. words

.

.R MYPROG

 ODT  V04.00
*
```

The following example is similar to the previous example, except that execution does not automatically begin with ODT. When you start the program (MYPROG in this case), you must specify the address of O.ODT as shown in the link map.

```
.LINK/MAP:TT: MYPROG,ODT
RT-11 LINK  V06.01      Load Map
MYPROG.SAV      Title:  MYPROG  Ident:  V04.00

Section  Addr   Size    Global  Value   Global  Value   Global  Value

. ABS.   000000 001000   (RW,I,GBL,ABS,OVR)
PROG     001000 002052   (RW,I,LCL,REL,CON)
                         START   001000
$ODT$    003052 006152   (RW,I,LCL,REL,CON)
                         O.ODT   003304

Transfer address = 003304, High limit = 011222 = 2377. words

.GET MYPROG

.START 3304

 ODT  V04.00
*
```

The next example links ODT with a bottom address of 4000, then loads ODT.SAV and MYPROG.SAV into memory. As in Example 3 above, when you start the program, you must specify the address of O.ODT as shown in the link map.

```
.LINK/MAP:TT: ODT/BOTTOM:4000
RT-11 LINK  V06.01      Load Map
ODT   .SAV      Title:  ODT     Ident:  V04.00          /B:004000

Section  Addr   Size    Global  Value   Global  Value   Global  Value

. ABS.   000000 004000   (RW,I,GBL,ABS,OVR)
$ODT$    004000 006152   (RW,I,LCL,REL,CON)
                        O.ODT   004232

Transfer address = 004232, High limit = 012150 = 2612. words

.GET ODT.SAV

.GET MYPROG.SAV

.START 004232

 ODT  V04.00
*
```

You can restart ODT by specifying O.ODT+2 as the start address. This reinitializes ODT and clears all breakpoints. For example:

```
.START 4234
*
```

You can reenter ODT by specifying O.ODT+4 as the start address. For example:

```
.START 4236

BE004242
*
```

If ODT is waiting for a command, a CTRL/C from the keyboard calls the keyboard monitor. The monitor responds with a ^C on the terminal and waits for a command. (You can use the REENTER command to reenter ODT only if your program has set the reenter bit and ODT is linked high in memory relative to the program; otherwise, ODT is reentered at address O.ODT+6.)

If you type CTRL/U during a search printout, the search terminates and ODT prints an asterisk.

21.2 Relocation

When the assembler produces a relocatable object module, the base address of the module is assumed to be location 000000. The addresses of all program locations, as shown in the assembly listing, are relative to this base address. After you link the module, many of the values and all of the addresses in the

program will be incremented by a constant whose value is the actual absolute base address of the module after it has been relocated. This constant is called the relocation bias for the module. Since a linked program may contain several relocated modules, each with its own relocation bias, and since, in the process of debugging, these biases will have to be subtracted from absolute addresses continually in order to relate relocated code to assembly listings, ODT provides automatic relocation.

The basis of automatic relocation is the eight relocation registers, numbered 0 through 7. You can set them to the values of the relocation biases at different times during debugging. Obtain relocation biases by consulting the link map. Once you set a relocation register, ODT uses it to relate relative addresses to absolute addresses. For more information on the relocation process, see Chapter 11.

ODT evaluates a relocatable expression as a 16-bit, six-digit (octal) number. You can type an expression in any one of the three forms presented in Table 16–1. In this table, the symbol n stands for an integer in the range 0 to 7 inclusive, and the symbol k stands for an octal number up to six digits long, with a maximum value of 177777. If you type more than six digits, ODT takes the last six digits typed, truncated to the low-order 16 bits. The symbol k may be preceded by a minus sign, in which case its value is the two's complement of the number typed. For example:

k (number typed)	Values
1	000001
–1	177777
400	000400
–177730	000050
1234567	034567

Table 21–1: Forms of Relocatable Expressions (r)

Form	Expression	Value of r
A	k	The value of *k*.
B	n,k	The value of *k* plus the contents of relocation register *n*. (If the *n* part of this expression is greater than 7, ODT uses only the last octal digit of *n*.)
C	C or C,k or n,C or C,C	Whenever you type the letter C, ODT replaces C with the contents of a special register called the constant register. (This value has the same role as the *k* or *n* that it replaces. The constant register is designated by the symbol $C and may be set to any value, as indicated below.)

Section 21.3.13 describes the relocation register commands in greater detail.

21.3 Commands and Functions

When ODT starts it indicates its readiness to accept commands by printing an asterisk at the left margin of the terminal. You can issue most of the ODT commands in response to the asterisk. You can examine a word and change it; you can run the object program in its entirety or in segments; you can search memory for specific words or references to them. The discussion below explains these features.

21.3.1 Printout Formats

Normally, when ODT prints addresses it attempts to print them in relative form (Form B in Table 21–1). ODT looks for the relocation register whose value is closest to, but less than or equal to, the address to be printed. It then represents the address relative to the contents of the relocation register. However, if no relocation register fits the requirement, the address prints in absolute form. Since the relocation registers are initialized to –1 (the highest number), the addresses initially print in absolute form. If you change the contents of any relocation register, it can then, depending on the command, qualify for relative form.

For example, suppose relocation registers 1 and 2 contain 1000 and 1004 respectively, and all other relocation registers contain much higher numbers. In this case, the following sequence might occur (the slash command causes the contents of the location to be printed; the line feed command, LF, accesses the next sequential location):

```
*1000;1R                    ;sets relocation register 1 to 1000
*1,4;2R                     ;sets relocation register 2 to 1004
*774/000000 <LF>            ;opens location 774
000776 /000000 <LF>         ;opens location 776
1,000000 /000000 <LF>       ;absolute location 1000
1,000002 /000000 <LF>       ;absolute location 1002
2,000000 /000000            ;absolute location 1004
```

The printout format is controlled by the format register, $F. Normally this register contains 0, in which case ODT prints relative addresses whenever possible. You can open $F and change its contents to a non-zero value, however. In that case all addresses will print in absolute form (see Section 21.3.4, Accessing Internal Registers).

21.3.2 Opening, Changing, and Closing Locations

An open location is one whose contents ODT prints for examination, making those contents available for change. In a closed location, the contents are no longer available for change. Several commands are used for opening and closing locations.

Any command (except for the slash and backslash commands) that opens a location when another location is already open causes the currently open location to be closed. You can change the contents of an open location by typing the new contents followed by a single-character command that requires no argument (that is, LF, ^, RET, ←, @, >, <).

21.3.2.1 Slash (/) — One way to open a location is to type its address followed by a slash. For example:

```
*1000/012746
```

This command opens location 1000 for examination and makes it ready to be changed.

If you do not want to change the contents of an open location, press the RETURN key to close the location. ODT prints an asterisk and waits for another command. However, to change the word, simply type the new contents before giving a command to close the location. For example:

```
*1000/012746  012345 <RET>
*
```

This command inserts the new value, 012345, in location 1000 and closes the location. ODT prints another asterisk, indicating its readiness to accept another command.

Used alone, the slash reopens the last location opened. For example:

```
*1000/012345  2340 <RET>
*/002340
```

This command opens location 1000, changes its address to 002340, and then closes the location. ODT prints an asterisk, indicating its readiness to accept another command. The / character reopens the last location opened and verifies its value.

Note again that opening a location while another is open automatically closes the currently open location before opening the new location.

Also note that if you specify an odd numbered address with a slash, ODT opens the location as a byte, and subsequently behaves as if you had typed a backslash (see Section 21.3.2.2).

21.3.2.2 Backslash (\) — ODT operates on bytes, as well as on words. Typing the address of the byte followed by a backslash character opens the byte. (On the LT33 or LT35 terminal, type SHIFT/L.) This causes ODT to print the byte value at the specified address, to interpret the value as ASCII code, and to print the corresponding character (if possible) on the terminal. (ODT prints a ? when it cannot interpret the ASCII value as a printable character.)

```
*1001\101 =A
```

A backslash typed alone reopens the last open byte. If a word was previously open, the backslash reopens its even byte:

```
*1002/000004  004 =?
```

21.3.2.3 LINE FEED Key (LF) — If you type the LINE FEED key when a location is open, ODT closes the open location and opens the next sequential location:

```
*1000/002340 <LF>
001002 /012740
```

In this example, the LINE FEED key caused ODT to print the address of the next location along with its contents and to wait for further instructions. After the above operation, location 1000 is closed and 1002 is open. You may modify the open location by typing the new contents.

If a byte location was open, typing a line feed opens the next byte location.

21.3.2.4 Circumflex or Up-Arrow (^) — If you type the circumflex (or up-arrow) when a location is open, ODT closes the open location and opens the previous location. (On LT33 or LT35, type SHIFT/N.) To continue from the example above:

```
*001002/012740 ^
001000 /002340
```

This command closes location 1002 and opens location 1000. You may modify the open location by typing the new contents.

If the opened location was a byte, then the circumflex opens the previous byte.

21.3.2.5 Underline or Back-Arrow (←) — If you type the underline (or back-arrow) (SHIFT/O on an LT33 or LT35 terminal) to an open word, ODT interprets the contents of the currently open word as an address indexed by the program counter (PC) and opens the addressed location:

```
*1006/000006 _
001016 /000405
```

Notice in this example that the open location, 1006, is indexed by the PC as if it were the operand of an instruction with addressing mode 67 (PC relative mode).

You can make a modification to the opened location before you type a line feed, circumflex, or underline. Also, the new contents of the location will be used for address calculations using the underline command. For example:

```
*100/000222 4<LF>          modifies to 4 and open next location
000102 /000111 6^          modifies to 6 and open previous location
000100 /000004 200_        changes to 200 and open location indexed
000302 /123456             by PC
```

21.3.2.6 Open the Addressed Location (@) — You can use the at (@) symbol (SHIFT/P on the LT33 or LT35 terminal) to optionally modify a location, close it, and then use its contents as the address of the location to open next. For example:

```
*1006/001044 @                ;opens location 1044 next
001044 /000500

*1006/001044 2100@            ;modifies to 2100 and opens location
002100 /000167                ;2100
```

21.3.2.7 Relative Branch Offset (>) — The right-angle bracket (>) optionally modifies a location, closes it, and then uses its low-order byte as a relative branch offset to the next word to be opened. For example:

```
*1032/000407  301>            ;modifies to 301 and interprets as a
000636 /000010                ;relative branch
```

Note that 301 is a negative offset (–77). ODT doubles the offset before it adds it to the PC; therefore, 1034 + (–176) = 636.

21.3.2.8 Return to Previous Sequence (<) — The left-angle bracket (<) lets you optionally modify a location, close it, and then open the next location of the previous sequence that was interrupted by an underline, @, or right-angle bracket command. Note that underline, @, or right-angle bracket causes a sequence change to the open word. If a sequence change has not occurred, the left-angle bracket simply opens the next location as a LINE FEED does. This command operates on both words and bytes.

```
*1032/000407  301>            ;> causes a sequence change
000636 /000010 <              ;returns to original sequence
001034 /001040 @              ;@ causes a sequence change
001040 /000405 \005 = <       ;< now operates on byte
001035 \002 =?  <             ;< acts like <LF>
001036 \004 =?
```

21.3.3 Accessing General Registers 0–7

Open the program's general registers 0–7 with a command in the following format:

$n/

The symbol, n, is an integer in the range 0–7 that represents the desired register. When you open these registers, you can examine them or change their contents by typing in new data, as with any addressable location. For example:

```
*$0/000033<RET>               ;examines register 0 then closes it
*

*$4/000474 464<RET>           ;opens register 4, changes its contents
*                             ;to 000464, then closes the register
```

The example above can be verified by typing a slash in response to ODT's asterisk:

```
*/000464
```

You can use the LINE FEED, circumflex, or @ command when a register is open.

21.3.4 Accessing Internal Registers

The program's status register contains the condition codes of the most recent operational results and the interrupt priority level of the object program. Open it by typing $S. For example:

```
*$S/000311
```

$S represents the address of the status register. In response to $S in the example above, ODT prints the 16-bit word, of which only the low-order eight bits are meaningful. Bits 0–3 indicate whether a carry, overflow, zero, or negative (in that order) has resulted, and bits 5–7 indicate the interrupt priority level (in the range 0–7) of the object program. (Refer to the *PDP–11 Processor Handbook* for the Status Register format.)

You can also use the $ to open certain other internal locations listed in Table 21–2.

Table 21–2: Internal Registers

Register	Section	Contents
$B	21.3.6	First word of the breakpoint table
$M	21.3.9	Mask location for specifying which bits are to be examined during a bit pattern search
$P	21.3.15	Defines the operating priority of ODT
$S	21.3.4	Condition codes (bits 0–3) and interrupt priority level (bits 5–7)
$C	21.3.10	Constant register
$R	21.3.13	Relocation register 0, the base of the Relocation Register table
$F	21.3.1	Format register

21.3.5 Radix–50 Mode (X)

Many PDP–11 system programs employ the Radix–50 mode of packing certain ASCII characters three to a word. You can use Radix–50 mode by specifying the MACRO .RAD50 directive. ODT provides a method for examining and changing memory words packed in this way with the X command.

When you open a word and type the X command, ODT converts the contents of the opened word to its three-character Radix–50 equivalent and prints these characters on the terminal. You can then type one of the responses from Table 21–3.

Table 21–3: Radix–50 Terminators

Response	Effect
RETURN key (RET)	Closes the currently open location
LINE FEED key (LF)	Closes the currently open location and opens the next one in sequence
Circumflex (^)	Closes the currently open location and opens the previous one in sequence
Any three characters whose octal code is 040 (space) or greater	Converts the three characters into packed Radix–50 format. Legal Radix–50 characters for this response are: . $ Space 0 through 9 A through Z

If you type any other characters, the resulting binary number is unspecified (that is, no error message prints and the result is unpredictable). You must type exactly three characters before ODT resumes its normal mode of operation. After you type the third character, the resulting binary number is available to be stored in the opened location. Do this by closing the location in any one of the ways listed in Table 21–3. For example:

```
*1000/042431 X=KBI CBA <RET>
*1000/011421 X=CBA
```

NOTE

After ODT converts the three characters to binary, the binary number can be interpreted in one of many different ways, depending on the command that follows. For example:

```
*1234/063337 X=PRO XIT/013704
```

Since the Radix–50 equivalent of XIT is 113574, the final slash in the example will cause ODT to open location 113574 if it is a legal address.

21.3.6 Breakpoints

The breakpoint feature helps you monitor the progress of program execution. You can set a breakpoint at any instruction that is not referenced by the program for data. When a breakpoint is set, ODT replaces the contents of the breakpoint location with a BPT trap instruction so that program execution is suspended when a breakpoint is encountered. Then the original contents of the breakpoint location are restored, and ODT regains control.

With ODT you can set up to eight breakpoints, numbered 0 through 7, at any one time. Set a breakpoint by typing the address of the desired location of the breakpoint followed by ;B. Thus, r;B sets the next available breakpoint at location *r*. (If all eight breakpoints have been set, ODT ignores the r;B command.) You may set or change specific breakpoints by the r;nB command, where n is the number of the breakpoint. For example:

```
*1020;B                    ;sets breakpoint 0
*1030;B                    ;sets breakpoint 1
*1040;B                    ;sets breakpoint 2
*1032;1B                   ;resets breakpoint 1 *
*
```

The ;B command removes all breakpoints. Use the ;nB command to remove only one of the breakpoints, where n is the number that identifies the breakpoint. For example:

```
*;2B                       ;removes breakpoint 2
*
```

ODT keeps a table of breakpoints that you can access. The $B/ command opens the location containing the address of breakpoint 0. The next seven locations contain the addresses of the other breakpoints in order. You can sequentially open them by using the LINE FEED key. For example:

```
*$B/001020 <LF>
001136 /001032 <LF>
001140 /007070 <LF>
001142 /007070 <LF>
001144 /007070 <LF>
001146 /001046 <LF>
001150 /001066 <LF>
001152 /007070
```

In this example, breakpoint 0 is set to 1020, breakpoint 1 is set to 1032, breakpoint 5 is set to 1046, and breakpoint 6 is set to 1066. The other breakpoints are not set.

Note that a repeat count in a proceed command (;P) refers only to the breakpoint that ODT most recently encountered. Execution of other breakpoints is determined by their own repeat counts. See Section 21.3.7, below.

21.3.7 Running the Program (r;G and r;P)

ODT controls program execution. There are two commands for running the program: r;G and r;P. The r;G command starts execution (go) and r;P continues (proceed) execution after halting at a breakpoint. For example:

```
*1000;G
```

This command starts execution at location 1000. The program runs until it encounters a breakpoint or until it completes. If it gets caught in an infinite loop, it must be either restarted or reentered as explained in Section 21.1.

Upon execution of either the r;G or r;P command, the general registers 0–6 are set to the values in the locations specified as $0–$6. The processor status register is set to the value in the location specified as $S.

When ODT encounters a breakpoint, execution stops and ODT prints Bn; (where n is the breakpoint number), followed by the address of the breakpoint. You can then examine locations for expected data. For example:

```
*1010;3B          ;sets breakpoint 3 at location 1010
*1000;G           ;starts execution at location 1000
B3;001010         ;stops execution at location 1010
*
```

To continue program execution from the breakpoint, type ;P in response to ODT's last prompt (*).

When you set a breakpoint in a loop, you can allow the program to execute a specified number of times through the loop before ODT recognizes the breakpoint. Set a proceed count by using the k;P command. This command specifies the number of times the breakpoint is to be encountered before ODT suspends program execution (on the kth encounter). The count *k* refers only to the numbered breakpoint that most recently occurred. You can specify a different proceed count for the breakpoint when it is encountered. Thus:

```
B3;001010         ;halts execution at breakpoint 3
*1026;3B          ;resets breakpoint 3 at location 1026
*4;P              ;sets proceed count to 4 and
B3;001026         ;continues execution; the program loops
*                 ;through the breakpoint three times and halts on
                  ;the fourth occurrence of the breakpoint
```

Following the table of breakpoints (as explained in Section 21.3.6) is a table of proceed command repeat counts for each breakpoint. You can inspect these repeat counts by typing $B/ and nine line feeds. The repeat count for breakpoint 0 prints (the first seven line feeds cause the table of breakpoints to be printed; the eighth types the single-instruction mode, explained in the next section, and the ninth line feed begins the table of proceed command repeat counts). The repeat counts for breakpoints 1 through 7 and the repeat count for the single-instruction trap follow in sequence. ODT initializes a proceed count to 0 before you assign it a value. After the command has been executed, it is set to –1. Opening any one of these provides an alternative way of changing the count. Once the location is open, you can modify its contents in the usual manner by typing the new contents followed by the RETURN key. For example:

```
.
.
.
nnnnnn /001036 <LF>     ;address of breakpoint 7
nnnnnn /006630 <LF>     ;single instruction address
nnnnnn /000000 15 <LF>  ;count for breakpoint 0; changes to 15
nnnnnn /000000 <LF>     ;count for breakpoint 1
```

```
.
.
.
nnnnnn  /000000 <LF>        ;count for breakpoint 7
nnnnnn /nnnnnn              ;repeat count for single instruction
                            ;mode.
```

Both the address indicated as the single-instruction address and the repeat count for single-instruction mode are explained in the following section.

21.3.8 Single-Instruction Mode

With this mode, you specify the number of instructions to be executed before ODT suspends the program run. The proceed command, instead of specifying a repeat count for a breakpoint encounter, specifies the number of succeeding instructions to be executed. Note that breakpoints are disabled in single-instruction mode. Table 21–4 lists the single-instruction mode commands.

Table 21–4: Single-Instruction Mode Commands

Command	Explanation
;nS	Enables single-instruction mode. (*n* can be any digit and serves only to distinguish this form from the form ;S, which disables single-instruction mode). Breakpoints are disabled.
n;P	Proceeds with program run for next n instructions before reentering ODT. (If *n* is missing, it is assumed to be 1.) Trapping instructions and associated handlers can affect the proceed repeat count (see Section 21.4.2).
;S	Disables single-instruction mode.

When the repeat count for single-instruction mode is exhausted and the program suspends execution, ODT prints:

```
B8; nnnnnn
```

where nnnnnn is the address of the next instruction to be executed. The $B breakpoint table contains this address following that of breakpoint 7. However, unlike the table entries for breakpoints 0–7, direct modification has no effect.

Similarly, following the repeat count for breakpoint 7 is the repeat count for single-instruction mode. You can modify this table entry directly. This is an alternative way of setting the single-instruction mode repeat count. In such a case, ;P implies the argument set in the $B repeat count table rather than an assumed 1.

21.3.9 Searches

With ODT you can search any specific portion of memory for bit patterns or references to a particular location.

21.3.9.1 Word Search (r;W) — Before initiating a word search, you must specify the mask and search limits. The location represented by $M specifies the mask of the search. $M/ opens the mask register. The next two sequential locations (opened by LINE FEEDs) initially contain the lower and upper limits of the search. ODT examines in the search all bits set to 1 in the mask and ignores other bits.

You must then give the search object and the initiating command, using the r;W command, where r is the search object. When ODT finds a match (that is, each bit set to 1 in the search object is set to 1 in the word ODT searches over the mask range), the matching word prints. For example:

```
*$M/ 000000 177400 <LF>          ;tests high-order eight bits
r,nnnnnn /000000  1000 <LF>      ;sets low address limit
r,nnnnnn /000000  1040 <RET>     ;sets high address limit
*400;W                           ;initiates word search
001010 /000770
001034 /000404
*
```

In the above example, *nnnnnn* is an address internal to ODT; this location varies and is meaningful only for reference purposes. In the first line above, the slash was used to open $M, which now contains 177400; the LINE FEEDs open the next two sequential locations, which now contain the upper and lower limits of the search.

In the search process, ODT performs an exclusive OR (XOR) with the word currently being examined and the search object; the result is ANDed to the mask. If this result is 0, a match has been found and ODT reports it on the terminal. Note that if the mask is 0, all locations within the limits print. This provides a convenient method for dumping all memory locations within given limits using ODT.

Typing CTRL/U during a search printout terminates the search.

21.3.9.2 Effective Address Search (r;E) — ODT provides a search for words that reference a specific location. Open the mask register only to gain access to the low- and high-limit registers. After specifying the search limits (as explained for the word search), type the command r;E (where *r* is the effective address) to initiate the search.

Words that are an absolute address (argument *r* itself), a relative address offset, or a relative branch to the effective address print after their addresses. For example:

```
*$M/ 177400 <LF>                 ;opens mask register only to gain
r,nnnnnn /001000 1010 <LF>       ;access to search limits
r,nnnnnn /001040 1060  <RET>
*1034;E                          ;initiates search
001016 /001006                   ;relative branch
001054 /002767                   ;relative branch
*1020;E                          ;initiates a new search
001022 /177774                   ;relative address offset
001030 /001020                   ;absolute address
```

Pay particular attention to the reported effective address references. A word can have the specified bit pattern of an effective address without actually being used as one. ODT reports all possible references whether they are actually used or not.

Typing CTRL/U during a search printout terminates the search.

21.3.10 Constant Register (rC)

It is often desirable to convert a relocatable address into its value after relocation, or to convert a number into its two's complement and then to store the converted value into one or more places in a program. Use the constant register to perform this and other useful functions.

Typing r;C evaluates the relocatable expression to its six-digit octal value, prints the value on the terminal, and stores it in the constant register. Invoke the contents of the constant register in subsequent relocatable expressions by typing the letter C. Examples follow:

```
*-4432;C=173346          ;places the two's complement of 4432 in the
                         ;constant register

*6632/062701 C <RET>     ;stores the contents of the constant
                         ;register in location 6632

*1000;1R                 ;sets relocation register 1 to 1000

*1,4272;C=005272         ;reprints relative location 4272 as an
                         ;absolute location and stores it in the
                         ;constant register
```

21.3.11 Memory Block Initialization (;F and ;I)

Use the constant register with the commands ;F and ;I to set a block of memory to a specific value. While the most common value required is 0, other possibilities are +1, –1, ASCII space, etc.

When you type the command ;F, ODT stores the contents of the constant register in successive memory words, starting at the memory word address you specify in the lower search limit and ending with the address you specify in the upper search limit.

Typing the command ;I stores the low-order eight bits in the constant register in successive bytes of memory, starting at the byte address you specify in the lower search limit and ending with the byte address you specify in the upper search limit.

For example, assume relocation register 1 contains 7000, 2 contains 10000, and 3 contains 15000. The following sequence sets word locations 7000–7776 to 0, and byte locations 10000–14777 to ASCII spaces:

```
*$M/000000 <LF>                  ;opens the mask register to gain
r,nnnnnn /000000 1,0 <LF>        ;access to search limits
r,nnnnnn /000000 2,-2 <RET>      ;sets the lower limit to 7000
*0;C=000000                      ;sets the upper limit to 7776
*;F                              ;sets the constant register to zero
                                 ;sets locations 7000–7776 to zero
*$M/000000 <LF>
r,nnnnnn /007000 2,0 <LF>        ;sets the lower limit to 10000
r,nnnnnn /007776 3,-1 <RET>      ;sets the upper limit to 14777
*40;C=000040                     ;sets the constant register to 40
                                 ;(space)
*;I                              ;sets the byte locations
*                                ;10000–14777
                                 ;to the value in the low-order
                                 ;eight bits of the constant
                                 ;register
```

21.3.12 Calculating Offsets (r;O)

Relative addressing and branching involve the use of an offset. An offset is the number of words or bytes forward or backward from the current location to the effective address. During the debugging session it is sometimes necessary to change a relative address or branch reference by replacing one instruction offset with another. ODT calculates the offsets in response to the r;O command.

The command r;O causes ODT to print the 16-bit and 8-bit offsets from the currently open location to address r. For example:

```
*346/000034 414;O 000044 022 22  <RET>
*/000022
```

This command opens location 346, calculates and prints the offsets from location 346 to location 414, changes the contents of location 346 to 22 (the 8-bit offset), and verifies the contents of location 346.

The 8-bit offset prints only if it is in the range –128(decimal) to 127(decimal) and the 16-bit offset is even, as was the case above. In the next example, the offset of a relative branch is calculated and modified so that it branches to itself.

```
*1034/103421 1034;O 177776 377 \021 =?  377<RET>
*/103777
```

Note that the modified low-order byte 377 must be combined with the unmodified high-order byte.

21.3.13 Relocation Register Commands

The use of the relocation registers is described briefly in Section 21.2. At the beginning of a debugging session it is desirable to preset the registers to the relocation biases of those relocatable modules that will be receiving the most attention. Do this by typing the relocation bias, followed by a semicolon and the specification of relocation registers, as follows:

r;nR

The symbol *r* may be any relocatable expression, and *n* is an integer in the range 0 – 7. If you omit *n*, it is assumed to be 0. For example:

```
*1000;5R                ;puts 1000 into relocation register 5
*5,100;5R               ;adds 100 to the contents
*                       ;of relocation register 5
```

Once a relocation register is defined, you can use it to reference relocatable values. For example:

```
*2000;1R                ;puts 2000 into relocation register 1
*1,2176/ 002466         ;examines the contents of location 4176
*1,3712;0B              ;sets a breakpoint at location 5712
```

Sometimes programs may be relocated to an address below the one at which they were assembled. This could occur with PIC code (position-independent code), which is moved without using the linker. In this case, the appropriate relocation bias would be the two's complement of the actual downward displacement. One method for easily evaluating the bias and putting it in the relocation register is illustrated in the following example.

Assume a program was assembled at location 5000 and was moved to location 1000. Then the following sequence enters the two's complement of 4000 in relocation register 1.

```
*1000;1R
*1,-5000;1R
*
```

Relocation registers are initialized to –1 so that unwanted relocation registers never enter into the selection process when ODT searches for the most appropriate register.

To set a relocation register to –1, type ;nR. To set all relocation registers to –1, type ;R.

ODT maintains a table of relocation registers, beginning at the address specified by $R. Opening $R ($R/) opens relocation register 0. Successively typing a LINE FEED opens the other relocation registers in sequence. When a relocation register is opened in this way, you can modify it as you would any other memory location.

21.3.14 The Relocation Calculators, n! and nR

When a location has been opened, it is often desirable to relate the relocated address and the contents of the location back to their relocatable values. To calculate the relocatable address of the opened location relative to a particular relocation bias, type:

```
n!
```

The symbol *n* specifies the relocation register. This calculator works with opened bytes and words. If you omit *n*, the relocation register whose contents are closest to, but less than or equal to, the opened location is selected automatically by ODT. In the following example, assume that these conditions are fulfilled by relocation register 3, which contains 2000. Use the following command to find the most likely module that a given opened byte is in:

```
*2500\011 = !=3,000500
```

To calculate the difference between the contents of the opened location and a relocation register, type:

```
nR
```

The symbol *n* represents the relocation register. If you omit *n*, ODT selects the relocation register whose contents are closest to, but less than or equal to, the contents of the opened location. For example, assume the relocation bias stored in relocation register 1 is 7000:

```
*1,500/011032 1R=1,002032
```

The value 2032 is the content of 1,500, relative to the base 7000. The next example shows the use of both relocation calculators.

If relocation register 1 contains 1000, and relocation register 2 contains 2000, use the following command to calculate the relocatable addresses of location 3000 and its contents, relative to 1000 and 2000:

```
*3000/006410 1!=1,002000 2!=2,001000 1R=1,005410 2R=2,00410
```

21.3.15 ODT Priority Level ($P)

$P represents a location in ODT that contains the interrupt (or processor) priority level at which ODT operates. If $P contains the value 377, ODT operates at the priority level of the processor at the time ODT is entered. Otherwise $P may contain a value between 0 and 7 corresponding to the fixed priority at which ODT operates.

To set ODT to the desired priority level, open $P. ODT prints the present contents, which you can then change:

```
*$P/000006 4 <RET>          ;lowers the priority to allow interrupts
*                           ;from the terminal
```

If you do not change $P, its value is seven.

You must set ODT's priority to 0 if you are using ODT in a foreground/background environment while another job is running.

ODT may not always service breakpoints that are set in routines that run at different priority levels. For example, a program running at a low priority can use a device service routine that operates at a higher priority level. If you set $P low, ODT waits for terminal input at a low priority. If an interrupt occurs from a high-priority routine, the breakpoints in the high-priority routine will not be recognized because they were removed when the earlier breakpoint occurred. Thus, interrupts that are set at a priority higher than the one at which ODT is running will be serviced, but any breakpoints will not be recognized. To avoid this problem, set breakpoints at one priority level at a time. That is, set breakpoints within an interrupt service routine, but not at mainline code level. For a more complete discussion of how the PDP–11 handles priority and interrupts, refer to the processor handbook for your particular machine. ODT disables all breakpoints in the program whenever it gains control. Breakpoints are enabled when ;P and ;G commands are executed. For example:

```
*$P/00007 5
*1000;B
*2000;B
*1000;G
B0;001000
*                                   ;an interrupt occurs and is serviced
```

If a higher-level interrupt occurs while ODT is waiting for input, the interrupt is serviced, and no breakpoints are recognized.

21.3.16 ASCII Input and Output (r;nA)

Inspect and change ASCII text by using a command of this syntax:

r;nA

The symbol *r* represents a relocatable expression, and *n* is a character count. If you omit *n*, it is assumed to be 1. ODT prints *n* characters starting at location *r* and followed by a carriage return/line feed combination. Table 21–5 lists responses and their effect.

Table 21–5: ASCII Terminators

Response	Effect
RETURN Key (<RET>)	ODT outputs a carriage return/line feed combination followed by an asterisk, and waits for another command.
LINE FEED Key (<LF>)	ODT opens the byte following the last byte that was output.
Up to *n* characters of text	ODT inserts the text into memory, starting at location *r*. If you type exactly *n* characters, ODT responds with *<CR><LF> address <CRXLF>**. If you type fewer than *n* characters, terminate that string with CTRL/U. ODT responds with *?^U <CR><LF> address <CR><LF>*.

21.4 Programming Considerations

Information in this section is not necessary for normal use of ODT. However, it does provide a better understanding of how ODT performs some of its functions. In certain difficult debugging situations, this understanding is necessary.

21.4.1 Using ODT with Foreground/Background Jobs

It is possible to use ODT to debug programs written as either background or foreground jobs. In the background or under the single-job monitor, you can link ODT with the program as described in Example 1 in Section 21.1. To debug a program in the foreground area, DIGITAL recommends that you run ODT in the background while the program to be debugged is in the foreground. The sequence of commands to do this is:

```
.FRUN PROG/P                ;loads the foreground program
LOADED AT nnnnnn            ;the first address of the job prints
.RUN ODT                    ;runs ODT in the background
ODT V01.01                  ;and sets a relocation register
* nnnnnn; ;OR               ;to the start of the job

*$F/000000 0                ;clears the format register to enable
*0,nnnnnn ;OB               ;proper address printing
                            ;sets a breakpoint

*0;G                        ;starts the keyboard monitor again

.RESUME                     ;starts the foreground job
```

The copy of ODT used must be linked low enough so that it fits in memory along with the foreground job.

NOTE

Since ODT uses its own terminal handler, it cannot be used with the display hardware. If GT ON is in effect, ODT ignores it and directs its input and output only to the console terminal.

If you use ODT in a foreground/background environment while another job is running, set ODT's priority bit to 0 as follows:

```
*$P/000007 0 <RET>
```

This puts ODT into the wait state at level 0, not at level 7. If you leave ODT's priority at 7, all interrupts (including clock) are locked out while ODT is waiting for terminal input.

21.4.2 Functional Organization

The internal organization of ODT is almost totally modularized into independent subroutines. The internal structure consists of three major functions: command decoding, command execution, and utility routines.

The command decoder interprets the individual commands, checks for command errors, saves input parameters for use in command execution, and sends control to the appropriate command execution routine.

The command execution routines take parameters saved by the command decoder and use the utility routines to execute the specified command. Command execution routines either return to the command decoder or transfer control to your program.

The utility routines are common routines such as SAVE-RESTORE and I/O. They are used by both the command decoder and the command executers.

21.4.3 Breakpoints

The function of a breakpoint is to give control to ODT whenever a program tries to execute the instruction at the selected address.

When a breakpoint is executed, ODT removes all the breakpoint instructions from the code so that you can examine and alter the locations. ODT then types a message on the terminal in the form Bn;r, where *r* is the breakpoint address and *n* is the breakpoint number. ODT restores the breakpoints when execution resumes.

There is a major restriction in the use of breakpoints: the program must not reference the word where a breakpoint was set since ODT altered the word. You should also avoid setting a breakpoint at the location of any instruction that clears the T-bit. For example:

```
MOV #240,177776      ;SET PRIORITY TO LEVEL 5
```

NOTE

> Instructions that cause traps or returns from them (for example, EMT, RTI) are likely to clear the T-bit, because a new word from the trap vector or the stack is loaded into the status register.

A breakpoint occurs when a trace trap instruction (placed in your program by ODT) is executed. When a breakpoint occurs, ODT operates according to the following algorithm:

1. Sets processor priority to seven (automatically set by trap instruction).
2. Saves registers and sets up stack.
3. If internal T-bit trap flag is set, goes to step 13.
4. Removes breakpoints.

5. Resets processor priority to ODT's priority or user's priority.
6. Makes sure a breakpoint or single-instruction mode caused the interrupt.
7. If the breakpoint did not cause the interrupt, goes to step 15.
8. Decrements repeat count.
9. Goes to step 18 if non-zero; otherwise resets count to one.
10. Saves terminal status.
11. Types message about the breakpoint or single-instruction mode interrupt.
12. Goes to command decoder.
13. Clears T-bit in stack and internal T-bit flag.
14. Jumps to the go processor.
15. Saves terminal status.
16. Types BE (bad entry), followed by the address.
17. Clears the T-bit, if set, in the user status and proceeds to the command decoder.
18. Goes to the proceed processor, bypassing the TT restore routine.

Note that steps 1–5 inclusive take approximately 100 microseconds. Interrupts are not permitted at this time, because ODT is running at priority level 7.

ODT processes a proceed (;P) command according to the following algorithm:

1. Checks the proceed for legality.
2. Sets the processor priority to seven.
3. Sets the T-bit flags (internal and user status).
4. Restores the user registers, status, and program counter.
5. Returns control to the user.
6. When the T-bit trap occurs, executes steps 1, 2, 3, 13, and 14 of the breakpoint sequence, restores breakpoints, and resumes normal program execution.

When a breakpoint is placed on an IOT, EMT, TRAP, or any instruction causing a trap, ODT follows this algorithm:

1. When the breakpoint occurs as described above, enters ODT.
2. When ;P is typed, sets the T-bit and executes the IOT, EMT, TRAP, or other trapping instruction.

3. Pushes the current PC and status (with the T-bit included) on the stack.

4. Obtains the new PC and status (no T-bit set) from the respective trap vector.

5. Executes the whole trap service routine without any breakpoints.

6. When an RTI is executed, restores the saved PC and PS (including the T-bit). Executes the instruction following the trap-causing instruction. If this instruction is not another trap-causing instruction, the T-bit trap occurs; reinserts the breakpoints in the user program, or decrements the single-instruction mode repeat count. If the following instruction is a trap-causing instruction, repeats this sequence starting at step 3.

NOTE

Exit from the trap handler must be by means of the RTI instruction. Otherwise, the T-bit is lost. ODT cannot regain control because the breakpoints have not yet been reinserted.

Note that the ;P command is illegal if a breakpoint has not occurred (ODT responds with *?*). ;P is legal, however, after any trace trap entry.

The internal breakpoint status words have the following format:

1. The first eight words contain the breakpoint addresses for breakpoints 0–7. (The ninth word contains the address of the next instruction to be executed in single-instruction mode.)

2. The next eight words contain the respective repeat counts. (The following word contains the repeat count for single-instruction mode.)

You may change these words at will, either by using the breakpoint commands or by directly manipulating $B.

When program runaway occurs (that is, when the program is no longer under ODT control, perhaps executing an unexpected part of the program where you did not place a breakpoint), give control to ODT by pressing the HALT key to stop the computer and then restarting ODT (see Section 21.1). ODT prints an asterisk, indicating that it is ready to accept a command.

If the program you are debugging uses the console terminal for input or output, the program can interact with ODT to cause an error because ODT uses the console terminal as well. This interactive error does not occur when you run the program without ODT.

Note the following rules concerning the ODT break routine:

1. If the console terminal interrupt is enabled upon entry to the ODT break routine, and no output interrupt is pending when ODT is entered, ODT generates an unexpected interrupt when returning control to the program.

2. If the interrupt of the console terminal reader (the keyboard) is enabled upon entry to the ODT break routine, and the program is expecting to

receive an interrupt to input a character, both the expected interrupt and the character are lost.

3. If the console terminal reader (keyboard) has just read a character into the reader data buffer when the ODT break routine is entered, the expected character in the reader data buffer is lost.

21.4.4 Searches

The word search lets you search for bit patterns in specified sections of memory. Using the $M/ command, specify a mask, a lower search limit ($M+2), and an upper search limit ($M+4). Specify the search object in the search command itself.

The word search compares selected bits (where 1's appear in the mask) in the word and search object. If all of the selected bits are equal, the unmasked word prints.

The search algorithm is:

1. Fetches a word at the current address.
2. XORs (exclusive OR) the word and search object.
3. ANDs the result of step 2 with the mask.
4. If the result of step 3 is zero, types the address of the unmasked word and its contents; otherwise, proceeds to step 5.
5. Adds two to the current address. If the current address is greater than the upper limit, types * and returns to the command decoder; otherwise, goes to step 1.

Note that if the mask is 0, ODT prints every word between the limits, since a match occurs every time (that is, the result of step 3 is always 0).

In the effective address search, ODT interprets every word in the search range as an instruction that is interrogated for a possible direct relationship to the search object. The mask register is opened only to gain access to the search limit registers.

The algorithm for the effective address search is as follows ((X) denotes contents of X, and K denotes the search object):

1. Fetches a word at the current address X.
2. If (X) = K [direct reference], prints contents and goes to step 5.
3. If (X) + X + 2 = K [indexed by PC], prints contents and goes to step 5.
4. If (X) is a relative branch to K, prints contents.
5. Adds 2 to the current address. If the current address is greater than the upper limit, performs a carriage return/line feed combination and returns to the command decoder; otherwise, goes to step 1.

21.4.5 Terminal Interrupt

When entering the TT SAVE routine, ODT follows these steps:

1. Saves the LSR status register (TKS).
2. Clears interrupt enable and maintenance bits in the TKS.
3. Saves the TT status register (TPS).
4. Clears interrupt enable and maintenance bits in the TPS.

To restore the TT:

1. Wait for completion of any I/O from ODT.
2. Restore the TKS.
3. Restore the TPS.

NOTE

If the TT printer interrupt is enabled upon entry to the ODT break routine, the following can occur:

1. If no output interrupt is pending when ODT is entered, an additional interrupt always occurs when ODT returns control to the user.
2. If an output interrupt is pending upon entry, the expected interrupt occurs when the user regains control.

If the TT reader (keyboard) is busy or done, the expected character in the reader data buffer is lost.

If the TT reader (keyboard) interrupt is enabled upon entry to the ODT break routine, and a character is pending, the interrupt (as well as the character) is lost.

21.5 Error Detection

ODT detects two types of error: illegal or unrecognizable command and bad breakpoint entry. ODT does not check for the legality of an address when you command it to open a location for examination or modification. Thus the command:

```
177774/
?MON-F-Trap to 4 003362
```

references nonexistent memory, thereby causing a trap through the vector at location 4. If the program you are debugging with ODT has requested traps through location 4 with the .TRPSET EMT, the program receives control at its TRPSET address.

If something other than a legal command is typed, ODT ignores the command and prints:

```
(echoes illegal command)?
*
```

ODT then waits for another command. Therefore, to cause ODT to ignore a command that has just been typed, type any illegal character (such as 9 or RUBOUT), and the command will be treated as an error and ignored.

ODT suspends program execution whenever it encounters a breakpoint (that is, traps to its breakpoint routine). If the breakpoint routine is entered and no known breakpoint caused the entry, ODT prints:

```
BE nnnnnn
*
```

and waits for another command. BE*nnnnnn* denotes bad entry from location nnnnnn. A bad entry may be caused by an illegal trace trap instruction, by a T-bit set in the status register, or by a jump to some random location within ODT.

Chapter 22
Save Image Patch Program (SIPP)

The Save Image Patch Program (SIPP) lets you make code modifications to any RT–11 file that exists on a random-access storage volume. You use SIPP primarily for maintaining save image files. Although SIPP is designed for maintaining programs that have been created with the RT–11 version 4 linker, you can use SIPP for pre-version 4 programs that are not overlaid.

SIPP is also useful for examining locations within a file. If you do not modify any locations within a file, SIPP makes no changes. SIPP's patching format is easier to use than that of PATCH (see Chapter 25). Also, you can run SIPP from an indirect command file, a BATCH stream, or from the console.

When you run SIPP, you have the option of installing your code modifications when you close the file, or you can create a command file that contains both the code modifications and the instructions necessary for SIPP to install them. You can run this command file as an indirect file whenever you wish.

Because SIPP does not install code modifications until you have finished making them, SIPP's checksum is not affected by a CTRL/U or DELETE. This feature also makes the code modification, or patching, procedure easier for you.

NOTE

DIGITAL does not recommend that you modify the following data within a save image file: locations 50, 64, and 66; the Job Status Word; the overlay handler; the overlay tables; and the window definition blocks. SIPP uses these locations for internal calculations and will automatically update them as necessary. Note, however, that if you use the /A option, SIPP does not modify any of these locations.

22.1 Calling and Using SIPP

To call SIPP, respond to the dot (.) printed by the keyboard monitor by typing:

```
R SIPP<RET>
```

The Command String Interpreter prints an asterisk (*) at the left margin of the terminal and waits for a command string. If you enter only a carriage return in response to the asterisk, SIPP prints its current version number. If you type a CTRL/C in response to the asterisk, control returns to the mon-

itor. If you type a CTRL/C in response to any of SIPP's prompts, SIPP prints the following confirmation message:

```
?SIPP -  Are you sure?
```

If you type a Y (or any string beginning with a Y) followed by a carriage return, SIPP aborts the patching procedure, and returns control to the monitor, without making any changes to your file. Any other response returns control to the procedure that was interrupted. You must type two consecutive CTRL/Cs at any other time, including while running from an indirect command file, to get the *?SIPP — Are you sure?* message.

Enter a command string according to this general syntax:

[com-filespec=]input-filespec[/option...]

where:

com-filespec	represents the file specifications of the command file that you want SIPP to create. You can run this file as an indirect file. The default file type is .COM. If you do not specify a command file, SIPP does not create one
input-filespec	represents the file specifications of the file you want to modify. If you do not specify a file type, SIPP assumes .SAV
/option	is one of the options listed in Table 22–1

If you enter only a device specification in response to the CSI asterisk, SIPP opens the first block of that volume and assumes the /A option.

22.2 SIPP Options

Table 22–1 summarizes the options that you can use in the CSI command string to SIPP.

Table 22–1: SIPP Options

Option	Function
/A	Prevents SIPP from automatically modifying either location 50, the window definition blocks, the overlay table, or the overlay handler. Use /A when you are patching anything other than save image files. When you use the option, SIPP modifies only those locations that you specify.
/C	Requires you to enter a checksum after you finish code modifications. If you make no modifications, SIPP ignores /C. The command file will automatically contain /C. You cannot use /C and /D together. See Section 22.5 for more details on the checksum.

(continued on next page)

Table 22–1: SIPP Options (Cont.)

Options	Function
/D	Use if you do not know the checksum for a particular patch and you want SIPP to create one. SIPP prints the checksum for the patch after you have finished entering all the code modifications. If you make no modifications, SIPP ignores the /D option. You cannot use /C and /D together.
/L	When you use /L, SIPP does not modify the input file after the patching session. This option is useful if you wish only to create a command file and preserve the input file.

22.3 SIPP Dialog

After you have entered the initial command string to SIPP, SIPP prints a series of prompts at the terminal. The responses you give to these prompts guide SIPP to the location in the input file or volume where you want to begin code modifications. If the input file is overlaid, the first prompt SIPP prints at the terminal is:

```
Segment?
```

Respond to this prompt by typing the number of the overlay segment that contains the locations you want to modify. (SIPP does not print this prompt: if the file you are modifying is not overlaid, if you are using the /A option, or if you are modifying a volume.) You can find the segment number in the program's load map. Type a carriage return, or 0 followed by a carriage return, if you want to modify the program's root segment.

SIPP prompts you for the base address within the program or overlay segment where you want to begin code modifications or examination. SIPP prints the following prompt for both overlaid and non-overlaid files. (Note that the following prompt is the second prompt for overlaid files, and the first prompt for anything else.)

```
Base?
```

If the file you are modifying is overlaid, respond to the last prompt by entering the base address specified on the load map for the segment you want to modify. If the file is not overlaid, enter the load address of the program section you wish to modify or examine.

After you have entered the base address, SIPP prompts you for the offset as follows:

```
Offset?
```

Respond to the offset prompt by typing the offset from the current base where you want to begin modifying or examining your program.

If the offset you specify is an even number, SIPP opens the corresponding location as a word. If the offset is odd, SIPP opens the location as a byte. Section 22.4 describes how you can alternate between words and bytes as you proceed to modify or examine the file.

After you have responded to *Offset?*, SIPP prints the following header:

```
Segment      Base      Offset      Old  New?
```

If the file is not overlaid, SIPP does not print the *Segment* column. Below the header, SIPP prints the segment, base, and offset you have specified by responding to the dialogue prompts.

A sample dialogue format follows. In this example, SIPP is to begin code modifications in overlay segment 2 of program PROG.SAV.

```
.R SIPP<RET>
*PROG=PROG<RET>
Segment? 2<RET>
Base?    20000<RET>
Offset?  100<RET>

Segment      Base      Offset      Old  New?
000002       20000       20100  103425
```

Under the column marked *Old*, SIPP prints the contents of the currently open location. Under the column designated *New?*, you can enter either a new value for the current location and/or a command. Section 22.4 gives more details on opening and modifying locations. Table 22–2 summarizes the commands you can enter.

NOTE

SIPP does not make changes to a file as you type them. Instead, SIPP stores the changes in a buffer, allowing you to abort a partially completed patch operation without leaving behind a partially patched file. When you finish a patching operation by typing CTRL/Y or multiple CTRL/Z's (see Table 22–2), SIPP makes all the changes in one pass.

22.4 SIPP Commands

Table 22–2 summarizes the commands you can enter during the code modification procedure and lists the sections in which you can find more details on each command. You can follow command with either a line feed or a carriage return.

Table 22–2: SIPP Commands

Command	Section	Function
<RET> or <LF>	22.4.1	Closes the current location without modifying it, and opens and displays the next location.
n<RET>	22.4.1	Enters the value represented by *n* in the current location, closes it, and opens the next location.
^<RET>	22.4.2	Closes the current location without modifying it, and opens the previous location.

(continued on next page)

Table 22–2: SIPP Commands (Cont.)

Command	Section	Function
n^<RET>	22.4.2	Enters the value represented by *n* in the current location, closes it, and opens the previous location.
\<RET>	22.4.3	Reopens the current location as a byte (starting with the low, or even, byte for that word). From this point, SIPP will continue opening byte locations and accepting byte values. Do not use this command when in Radix–50 or ASCII mode.
/<RET>	22.4.3	Reopens the current location as a word. SIPP displays the contents of the currently open word location. All further displays and input will be word values. Do not use this command when in Radix–50 or ASCII mode.
;O<RET>	22.4.4	Reopens the current location as an octal word value. This is the default mode. Use ;O to return to octal after having been in Radix–50 or ASCII mode. All further displays and input are octal values.
;A<RET>	22.4.5	Displays the byte of the current location as an ASCII value. All further displays are in ASCII, and SIPP advances in byte mode.
;Ax<RET>	22.4.5	Inserts an ASCII character represented by *x* in the byte of the current location, closes that byte, and opens and displays the next location. Use this command for inserting only one ASCII character at a time. You can also use this command to search for an ASCII value (see Section 22.4.7).
;R<RET>	22.4.6	Displays the current location as a word of up to three Radix–50 characters. All further displays are in Radix–50.
;Ryyy<RET>	22.4.6	Inserts up to three Radix–50 characters represented by *yyy* into the current location. SIPP then closes the current location, and opens and displays the next location. Use this command for inserting up to three Radix–50 characters. You can also use this command to search for a Radix–50 value (see Section 22.4.7).
;S<RET>	22.4.7	Searches for a value within the file. When you type this command, SIPP prompts you for a value for which it is to search. SIPP also prompts you for the boundaries within which you want it to conduct the search.
;V<RET>	22.4.8	Prints all the modifications you have made in the current patching session. You can use this command at any time, except in response to *Checksum?*.
CTRL/Z<RET>	22.4.10	Backs up to the previous prompt: *Offset?*, *Base?*, or *Segment?*. This command allows you to insert code modifications in more than one area of the file during the same patching session.
CTRL/Y<RET>	22.4.9	Completes the current patching session, installs the patch, creates the command file (if requested), and prompts you with an asterisk for another file specification.

22.4.1 Opening and Modifying Locations Within a File

As stated earlier, after you guide SIPP to the location in the file where you want to begin making code modifications, SIPP prints out a header under which it lists the address and contents of the location specified, called the current location. Under the last column in the header, *New?*, you can enter a value that replaces the contents of the current location. After you enter the new value, type a carriage return to advance to the next location.

In the following example, the value 240 is inserted in the current location. Next, a carriage return advances SIPP to the next 16-bit location.

```
Base        Offset        Old    New?
 001000      001200     004767   240<RET>
 001000      001202     003106
```

If you do not want to modify the current location, simply type a carriage return to advance to the next location.

22.4.2 Backing Up Through Files

When you type the uparrow character (^) followed by a carriage return in place of entering data into the current location, SIPP closes the current location and opens the previous location. If you specify a value followed by an uparrow, SIPP enters that value into the current location, closes that location, and opens the previous location.

If you type an uparrow when the offset from a specified base is 0, SIPP opens the previous location and displays the offset as a double-precision negative number.

In the following example, the value 112000 is entered into the current location, and the previous location is opened.

```
Base        Offset        Old    New?
 002000      002134     020027   112000^<RET>
 002000      002132     001732
```

In the last example, notice how SIPP decrements the offset by 2 to designate the previous 16-bit location.

You can use the uparrow with the backslash (\) to back up to the previous byte (if currently in word mode). You can also use the uparrow with the slash (/) to back up to the previous word (if currently in word mode).

22.4.3 Advancing in Bytes

By default, SIPP operates in word mode. That is, locations are displayed as 16-bit words, and values are displayed and entered as word values. If you type the backslash character (\), SIPP closes the current location, and

reopens the low, or even, byte of that location. Values that are displayed and entered from this point are in bytes.

To revert to word mode, type the slash character (/). When you type the slash, SIPP reopens the current location as a 16-bit word.

The following example uses the backslash to advance in byte mode, and then the slash to revert to word mode. Notice that SIPP prints out a new header each time it changes from word to byte mode, and vice versa.

```
Base        Offset        Old   New?
 002000      002112     003002  \<RET>

Base        Offset      Old     New?
 002000      002112     002     <RET>
 002000      002113     006     <RET>
 002000      002114     132     /<RET>

Base        Offset        Old   New?
 002000      002114     003132
```

If you are in byte mode when you get the *Offset?* prompt, SIPP automatically resets itself to word mode.

22.4.4 Entering Octal Values (;O)

Use the ;O command, followed by a carriage return, to reopen the current location, and display its contents as an octal value. Since octal mode is the default setting, you need to use it only if you are currently operating in ASCII or Radix–50 mode and wish to revert to octal mode. You can also use the ;O command to switch from byte mode to word mode.

The following example uses the ;O command to switch from ASCII mode to octal mode.

```
Base        Offset        Old   New?
 002000      002100     051101  ;A<RET>
 002000      002100       <A>   ;O<RET>

Base        Offset        Old   New?
 002000      002100     051101
```

Note that unlike the ;A and ;R commands ;O accepts no optional argument. If you return to the *Offset?* prompt, SIPP automatically resets itself to octal mode.

22.4.5 Displaying and Entering ASCII Values

Use the ;A command, followed by a carriage return, to open the current location as a byte and display its contents as an ASCII value. When you use the ;A command, SIPP continues to display contents in ASCII until you use the ;O or ;R commands. Note that when you operate in ASCII mode, you advance through the file in byte mode.

The following example uses the ;A command to open the low byte of the current location and display its contents as an ASCII value.

```
Base        Offset       Old   New?
 003000      003100   050524   ;A<RET>
 003000      003100      <T>   <RET>
 003000      003101      <Q>
```

Use the ;Ax command to insert an ASCII character, represented by *x*, into the low byte of the current location. When you use the ;Ax command, SIPP enters the ASCII character directly into the current byte, closes that byte, opens and displays the next location as an octal, ASCII, or Radix–50 value (depending on what mode you were in prior to using the ;Ax command). Note that you can insert only one ASCII character at a time, and that you should not insert control characters. You can use the ;Ax command when displaying in ASCII mode.

The next example uses the ;Ax command to enter the ASCII character, W, into the current byte and proceed to the next byte.

```
Base        Offset       Old   New
 003000      003100   050524   ;AW<RET>
 003000      003101      121
```

You can also use the ;Ax command to search for an ASCII value (see Section 22.4.7).

22.4.6 Displaying and Entering Radix–50 Values

Use the ;R command, followed by a carriage return, to reopen the current location and display its contents in Radix–50. When you use the ;R command, SIPP continues in Radix–50 mode until you use either the ;A or ;O commands. Note that Radix–50 mode advances in word mode.

The following example uses the ;R command to reopen the current location and display its contents as a Radix–50 value.

```
Base        Offset       Old   New?
 001000      005220   071070   ;R<RET>
 001000      005220     <RK>   <RET>
 001000      005222    <TES>
```

If the contents of a location is an invalid Radix–50 value, SIPP displays the contents as <*???*>.

You can use the ;Ryyy command to insert up to three Radix–50 characters, represented by *yyy*, into the current location. When you use the ;Ryyy command, SIPP inserts the Radix–50 value into the current location, closes the current location, and opens and displays the contents of the next location as an octal, ASCII, or Radix–50 value (depending on what mode you were in prior to using the ;Ryyy command).

If you use the ;Ryyy command, and you enter only two Radix–50 characters, SIPP inserts a blank as the third character. Likewise, if you enter only one Radix–50 character, SIPP inserts blanks for the second and third characters. If you use an imbedded blank (for example, X Z), SIPP inserts all characters as typed. Note that you can insert up to only three Radix–50 characters at a time. Use the ;Ryyy command only when the low byte of the current location is open; SIPP prints an error message if you attempt to insert a Radix–50 value when the high byte of the current location is open.

The following Radix–50 values are valid for use with the ;Ryyy command:

A through Z
0 through 9
$
*
.
%

Note that a space is also a valid Radix–50 character, and that SIPP translates the percentage character to a dot (.).

The following example uses the ;Ryyy command to insert three Radix–50 characters in the current location, and proceed to the next location.

```
Base        Offset       Old   New?
 001000      005332   000240   ;RABC<RET>
 001000      005334   002110
```

You can also use the ;Ryyy command to search for a Radix–50 value (see Section 22.4.7).

22.4.7 Searching Through Files (;S)

You can use the ;S command to search between two specified boundaries of a file for a given value. With this feature, you can find the location where you want to make a change by searching for a specific value.

To request a search, type the following in response to any of SIPP's dialogue questions or in place of entering new data into the current location.

```
;S
```

Do not type the ;S command in response to the *Checksum?* prompt. After you type the ;S command, SIPP responds with the following prompt:

```
Search for?
```

Enter the value for which you want SIPP to search. You can use the ;Ax or ;Ryyy commands in response to the last prompt to search for ASCII or Radix–50 values. If you type a backslash after the value you enter, SIPP searches for a byte value. Otherwise, it searches for a word value. Note that if you use the ;Ax notation to search for an ASCII value, SIPP conducts the search in byte mode.

SIPP then asks for the lower address limit at which to begin the search:

```
Start?
```

Enter an address, followed by a carriage return, or just a carriage return. If you enter a carriage return, SIPP begins its search at the beginning of the file. If you enter an address, SIPP begins the search at that address. If you are searching through an overlay segment, use the following notation for the start address:

```
n:m
```

In the *n:m* notation, *n* represents the number of the segment you want to search, and *m* represents the offset from the start of the segment where you want SIPP to begin the search.

SIPP then asks for the upper address limit for the search:

```
End?
```

You can enter an address (including the n:m notation) or a carriage return. If you enter a carriage return, SIPP searches to the end of the file or volume. (If you use the /A option, SIPP searches to the end of the last block in the file or volume, otherwise it searches up to and including the last address in the program.) If you enter an address, SIPP conducts the search up to, but not including, that address.

After you have specified the search limits, the search begins. Each time SIPP finds a value that matches the one you specified, SIPP prints out the address of that value each time the value is found. If you use the /A option or if SIPP is searching the root segment of a program, SIPP prints the search results as follows:

```
Found at nnnnnn
```

If a search crosses segments in an overlaid file, SIPP prints the following each time it finds the specified value:

```
Found at seg:mmm,nnn
```

In the *seg:mmm,nnn* notation, *seg* represents the segment number, *mmm* represents the load map address of the segment, and *nnn* represents the offset from the start of the specified segment. Note that if you are searching an overlaid file, and you have specified the /A option in the command line, SIPP does not use the *seg:mmm,nnn* notation.

22.4.8 Verifying (;V)

Use the ;V command to list at the terminal all the changes you have made during the current patching session After SIPP prints out the addresses and new contents of all the locations that have changed, SIPP returns you to the operation that was interrupted.

You can use the ;V command at any time, except in response to the *Checksum?* prompt and the search *Start?*, and *End?* prompts. You can use the ;V command in response to the *Search for?* prompt.

The following example uses the ;V command to list at the terminal all the changes that have been made during the current patching session.

```
Base        Offset        Old   New
 003000      003200     003112  240<RET>
 003000      003202     002300  <RET>
 003000      003204     002300  <RET>
 003000      003206     000230  240<RET>
 003000      003210     000101  ;V<RET>

Base        Offset        Old   New?
 003000      003200     003112  000240
 003000      003206     000230  000240

Base        Offset        Old   New?
 003000      003210     000101
```

Note that when you use the ;V command to verify your modifications, all displays are in octal words. Note also that if you change a location and later restore that location to its original contents, SIPP includes that location in the verification.

22.4.9 Backing Up to a Previous Prompt

You can use the CTRL/Z sequence (or uparrow-Z), followed by a carriage return, to back up to a previous prompt. For example, after you have modified a series of locations, you can type CTRL/Z, followed by a carriage return, to back up to the *Offset?* prompt. Backing up to a previous prompt enables you to examine and/or modify other series of locations in your program.

If you use CTRL/Z in place of entering a value into a location, SIPP prompts *Offset?*. If you type CTRL/Z, followed by carriage return, in response to *Offset?*, SIPP prompts *Base?*. If you type yet another CTRL/Z, followed by a carriage return, SIPP either:

1. prompts *Segment?*, if the file is overlaid, or
2. prompts you for a checksum (if you used /C), then installs the patch (if the checksum is valid). (Note that if you have used the /L option, SIPP does not install the patch, but does create the command file, if requested.)

If the file is overlaid, and you type another CTRL/Z followed by a carriage return sequence in response to *Segment?*, SIPP prompts you for a checksum (if specified) then installs the modifications.

Using CTRL/Y provides a more efficient way of installing a patch (see the following subsection). The CTRL/Z sequence is designed primarily to request a particular prompt.

22.4.10 Completing Code Modifications

You can type the CTRL/Y sequence (or uparrow-Y), followed by a carriage return, to install the code modifications you have entered. If you have used the /C option, which requires you to enter a checksum, SIPP will prompt you for a checksum before it installs the modifications. If the checksum you type is valid, SIPP then installs the patch. If you have used the /L option and you enter the correct checksum, SIPP does not install the patch, but does create the command file, if requested.

After SIPP installs the modifications, an asterisk appears in the left margin, indicating that SIPP is ready to accept a new command string.

22.4.11 Extending Files and Overlay Segments

The limits to which you can extend programs and overlay segments while patching vary, depending on if the program is

- non-overlaid
- overlaid, but has only low memory overlays
- overlaid, but has only extended memory overlays
- overlaid, and has both low memory and extended memory overlays

The subsections that follow describe in detail the restrictions on extending programs, root sections, and overlay segments. Each subsection also details what data within your program SIPP does or does not automatically modify as you extend root sections and/or overlay segments.

Listed below are the data that SIPP may automatically modify as you make extensions. (Note that each is identified by an abbreviation; the subsections that follow reference these data by their abbreviations.)

Location 50	Contains the last address used by the program, if program is non-overlaid. If the program has low memory overlays, location 50 contains the last address used by the low memory overlay region(s). If the program has only extended memory overlays, location 50 contains the last address used by the root.
Reg. Size	Indicates the size of the extended memory region. This data appears in the extended memory overlay handler.
High Root + 2	Indicates the address of the next available location beyond the root segment. This data appears in either the low memory overlay handler or the extended memory handler.

High /O + 2 — Indicates the address of the next available location beyond the last low memory overlay region. This data appears in either the low memory overlay handler or the extended memory overlay handler.

Wdcnt. in Seg. — Indicates the number of words in the current overlay segment. This data appears in the overlay handler segment table.

WDB Size and Length — Indicates the window size and length to map in the window definition block (WDB) for the extended memory overlay you are extending. This data appears in each extended memory overlay segment's WDB.

WDB Offset — Indicates the offset into the extended memory region of the windows following the segment you are extending. This data appears in each extended memory overlay segment's WDB.

22.4.11.1 Non-overlaid Program — If necessary, SIPP automatically extends a non-overlaid program up to the end of the last block of the save image on which the program exists (SIPP automatically modifies location 50). Refer to DUP (Chapter 8) for details on extending a non-overlaid program beyond that point.

22.4.11.2 Overlaid Program, Low Memory Overlays Only — If your program is overlaid, but has only low memory overlays, SIPP does not permit you to extend the root. If an overlay segment is not in the last overlay region, you can extend it to the size of the largest segment in its region. If the overlay segment is in the last overlay region, you can extend it to the end of the last block of that overlay segment.

Table 22–3 shows the locations that SIPP modifies if necessary when you extend the root or overlay segment of a program that has low memory overlays only. Note in this table that there is a column heading for an overlay segment that is not the largest in its overlay region (*Not Largest in Region*), and for an overlay segment that you extend beyond the largest overlay segment in its region (*Past Largest in Last Region*). *YES* indicates that SIPP does modify the data in question if necessary, *NO* indicates SIPP does not modify the data in question, and *N/A* indicates that the data in question is not applicable.

Table 22–3: Overlaid Program Segment Limits

	Not Largest in Region	Past Largest in Last Region
Location 50	NO	YES
Reg. Size	N/A	N/A
High Root + 2	NO	NO
High /O + 2	NO	YES
Wdcnt. in Seg.	YES	YES
WDB Size and Length	N/A	N/A
WDB Offset	N/A	N/A

22.4.11.3 Overlaid Program, Extended Memory Overlays Only — If your program is overlaid, but has extended memory overlays only, you can extend the root segment up to the end of the last block on which the root resides. You can also extend any overlay segment up to the end of the last block of that particular segment, so long as you do not exceed the physical address space.

Table 22–4 shows the locations that SIPP modifies if necessary when you extend the root or overlay segment of a program that has extended memory overlays only. Note in this table that there is a column heading for the root (*Root*), an overlay segment that is not the largest in its overlay region (*Not Largest*), and an overlay segment that you extend beyond the largest overlay segment in its region (*Past Largest*). YES indicates that SIPP does modify the data in question if necessary, and NO indicates SIPP does not modify the data in question.

Table 22–4: Overlaid Program Segment Limits

	Root	Not Largest	Past Largest
Location 50	YES	NO	NO
Reg. Size	NO	YES	YES
High Root + 2	YES	NO	NO
High /O + 2	YES	NO	NO
Wdcnt. in Seg.	NO	YES	YES
WDB Size and Length	NO	YES	YES
WDB Offset	NO	YES	YES

22.4.11.4 Overlaid Program, Both Low Memory and Extended Memory Overlays — If your program has both low memory and extended memory overlays, SIPP does not permit you to extend the root segment. You can extend any low memory overlay segment up to the size of the largest segment in the same region. If the low memory overlay segment is in the last low memory overlay region, you can extend it to the end of the last block of that overlay segment.

You can extend any extended memory overlay segment up to the end of the block limit of that particular segment, so long as you do not exceed your physical address space.

Table 22–5 shows the locations that SIPP modifies if necessary when you extend the root or overlay segment of a program that has both low memory and extended memory overlays. Note in this table that there is a column heading for a low memory overlay segment that is not the largest in its overlay region (*/O Not Largest*), a low memory overlay segment that extends beyond the largest segment in its region (*/O Past Largest*), an extended memory overlay segment that is not the largest in its region (*/V Not Largest*), and an extended memory overlay segment that extends beyond the largest segment in its region (*/V Past Largest*). *YES* indicates that SIPP does modify the data in question if necessary, and *NO* indicates SIPP does not modify the data in question.

Table 22–5: Overlaid Program Segment Limits

	/O Not Largest	/O Past Largest	/V Not Largest	/V Past Largest
Location 50	NO	YES	NO	NO
Reg. Size	NO	NO	YES	YES
High Root + 2	NO	NO	NO	NO
High /O + 2	NO	YES	NO	NO
Wdcnt. in Seg.	YES	YES	YES	YES
WDB Size and Length	NO	NO	YES	YES
WDB Offset	NO	NO	YES	YES

NOTE

It is possible when extending extended memory overlay segments to exceed the 96K word physical address space (SIPP prints a warning message if you do this). If you do exceed the 96K word limit, use the CTRL/C sequence to abort the patching session; many system communication area locations will have already changed to contain invalid data.

22.5 SIPP Checksum

SIPP's checksum algorithm creates the checksum only after you have finished creating a patch. The checksum helps you verify your work. It lets you compare the patch you make to another that is known to be correct. The checksum does not tell you where your error is, but it does tell you that an inconsistency exists. SIPP's checksum algorithm uses the calculated address of a changed location and its new contents. SIPP includes in its checksum only those values of locations that have changed during a patching session.

If you change a word several times during a patching session, SIPP enters into its checksum only the last value you specify. This feature allows you to correct a mistake, yet maintain a valid checksum.

If you are creating a checksum (/D option), SIPP prints the following when you finish the patch:

```
Checksum=nnnnnn
```

If you are verifying a checksum (/C option), SIPP asks the following when you complete the patch:

```
Checksum?
```

Respond by entering the checksum for the patch.

If the checksum is incorrect, SIPP prints:

```
?SIPP-F-Checksum error
```

SIPP then returns to the beginning of its dialogue (the *Segment?* or *Base?* prompt), allowing you to find and correct your error without exiting from the patching session. In this way, you do not lose the changes you have made; you can go back and verify them (see Section 22.4.8 for details on the verify command).

SIPP does not install the patch until you enter the correct checksum. You can type CTRL/C to abort the patching procedure.

22.6 Running SIPP from an Indirect File

The SIPP indirect command file contains the commands necessary to install a patch in a particular file or volume. The order in which the modifications appear in the command file may not correspond to the actual sequence in which you typed them; however, the changes are the same as you typed. The contents of the command file always appear as octal word values. When you specify a command file in the initial command string, SIPP creates that file for use as an indirect command file. If you use the /L option when you create the command file, SIPP installs the patch contained within only when you run this file as an indirect command file. By default, SIPP assigns this file a .COM default file type.

A command file always contains a checksum generated during the console input session. If you use the /C option, SIPP prompts you for a checksum after you finish making the code modifications. If the checksum is valid, SIPP completes this command file, and you can execute this command file at any time you wish. If you use the /A option, SIPP inserts /A in the command file.

The command file TEST.COM is created in the following example. (Note that SIPP does not modify TEST.SAV in the example, because /L was specified in the command string.)

```
.R SIPP<RET>
*TEST=TEST/L<RET>
Base?   5000<RET>
Offset? 20<RET>

Base        Offset        Old   New?
 005000      000020    032764  240<RET>
 005000      000022    177400  240<RET>
 005000      000024    000002  1016<RET>
 005000      000026    001016  ^Y<RET>
 *^C
```

A copy of TEST.COM as it appears in indirect command file format follows.

```
.TYPE TEST.COM
RUN SIPP
DK:TEST.SAV/C
5000
20
240
240
1016
^Y
165617
^C
```

The number 165617 (the last line in the file) is the checksum for that patch.

To run the command file TEST.COM as an indirect file, type the following in response to the monitor dot.

```
@TEST<RET>
```

If you run a SIPP indirect command file when the SET TT: QUIET setting is in effect, SIPP overstrikes its output at the terminal but does install the patch correctly.

22.7 Running SIPP from a BATCH Stream

An easy way to install a patch from a BATCH stream is to follow the instructions for creating a command file. When you get your command file, simply open it with an editor, enter the BATCH commands, and insert a dot before the line RUN SIPP, and insert asterisks before each subsequent line. Remember to remove the CTRL/C (^C) from the command file. An example of preparing TEST.COM (from the previous section) for a BATCH stream follows.

```
$JOB/RT11
        TTYIO
.RUN SIPP
*DK:TEST.SAV/C
*5000
*20
*240
*240
*1016
*^Y
*165617
$EOJ
```

Chapter 23
Object Module Patch Utility (PAT)

The RT–11 Object Module Patch Utility (PAT) allows you to update code in a relocatable binary object module (.OBJ file). PAT does not permit you to examine the octal contents of an object module; use PATCH (described in Chapter 25) to do that. PAT makes the patch to the object module by means of the procedure outlined in Figure 23–2. One advantage to using PAT is that you can add relatively large patches to an object module without performing any octal calculations. PAT accepts a file containing corrections or additional instructions and applies these corrections and additions to the original object module. You prepare correction input in source form and assemble it with the MACRO–11 assembler.

Two files form the input to PAT: (1) the original input file, and (2) a correction file containing the corrections and additions to that input file. The original input file consists of one or more concatenated object modules, only one of which can be corrected with a single execution of the PAT utility. The correction file consists of object code that, when linked by the linker, either replaces or appends to the original object module. Output from PAT is the updated input file.

It is always good practice to create a backup version of the file you want to patch before you use PAT to make the changes.

23.1 Calling and Using PAT

To call PAT from the system device, respond to the dot (.) printed by the keyboard monitor by typing:

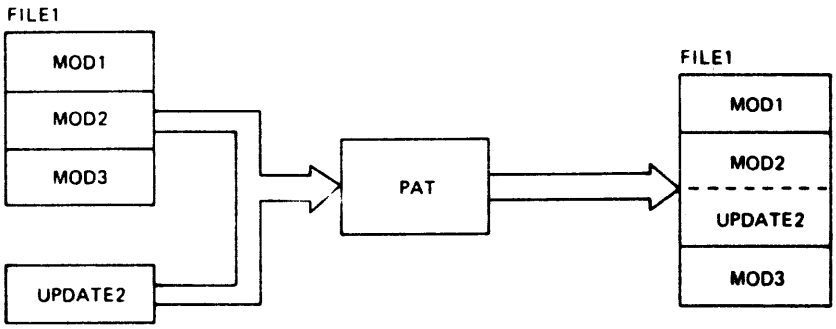
```
R PAT <RET>
```

The Command String Interpreter prints an asterisk at the left margin of the console terminal when it is ready to accept a command line. Chapter 6 describes the general syntax of the command line PAT accepts.

Type two CTRL/Cs to halt PAT at any time (or a single CTRL/C to halt PAT when it is waiting for console terminal input) and return control to the monitor. To restart PAT, type R PAT in response to the monitor's dot. When PAT executes an operation it returns control to the RT–11 monitor.

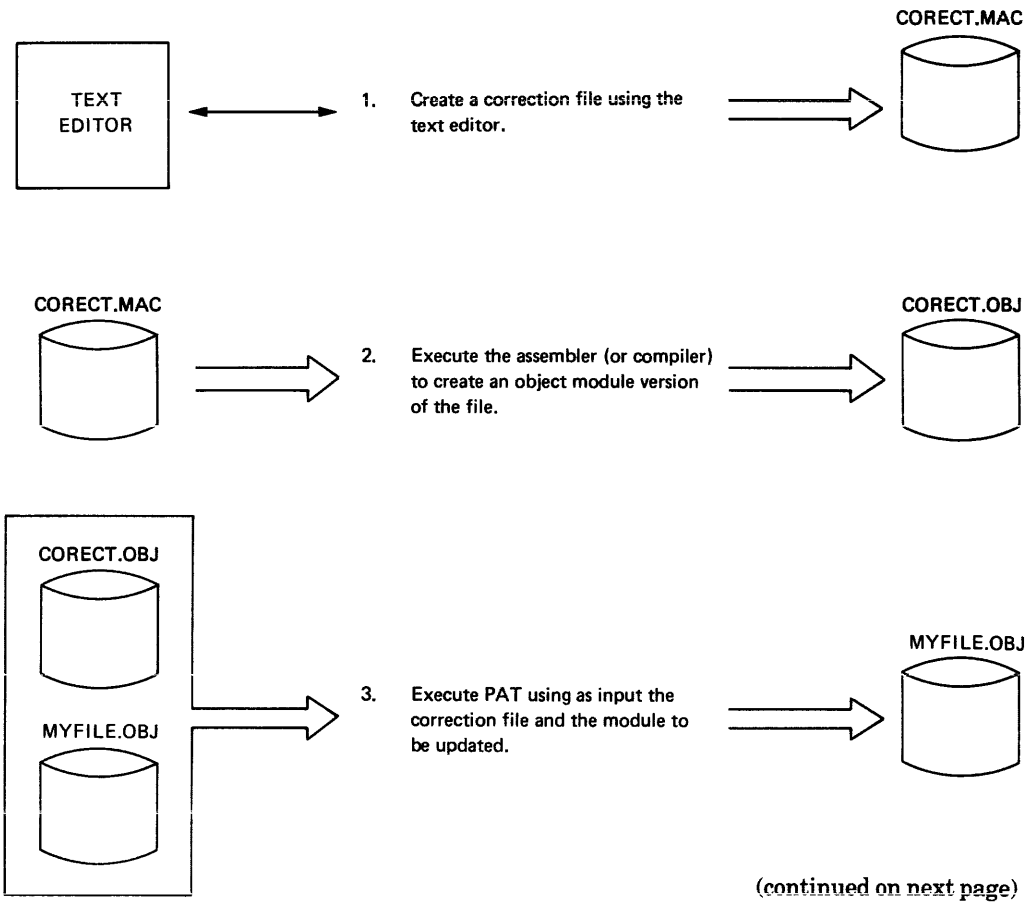
Figure 23–1 shows how you use PAT to update a file (FILE1) consisting of three object modules (MOD1, MOD2, and MOD3) by appending a correction file to MOD2. After running PAT, you use the linker to relink the updated module with the rest of the file and to produce a corrected executable program.

Figure 23–1: Updating a Module Using PAT

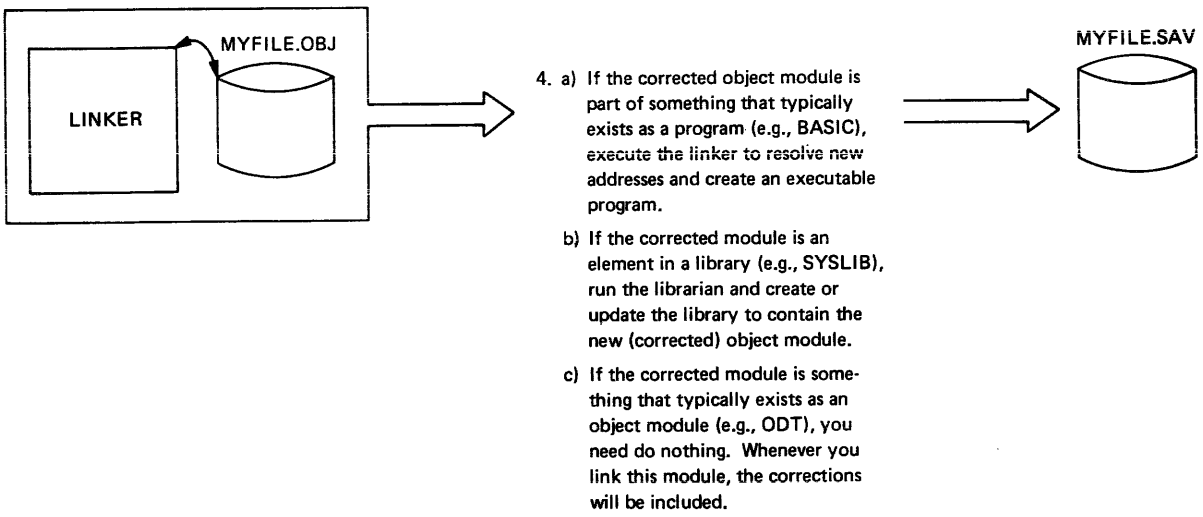


There are several steps you must follow when using PAT to update a file. First, use a text editor to create the correction file. Then, assemble the correction file to produce an object module. Next, submit the input file and the correction file in object module form to PAT for processing. Finally, link the updated object module, along with the object modules that make up the rest of the file, to resolve global symbols and create an executable program. Figure 23–2 shows the processing steps involved in generating an updated executable file using PAT.

Figure 23–2: Processing Steps Required to Update a Module Using PAT



(continued on next page)



Specify the PAT command string in the following form:

[output-filespec]=input-filespec[/C[:n]],correct-filespec[/C[:n]]

where:

output-filespec	is the file specification for the output file. If you do not specify an output file, PAT does not generate one
input-filespec	is the file specification for the input file. This file can contain one or more concatenated object modules
correct-filespec	is the file specification for the correction file. This file contains the updates being made to a single module in the input file
C	specifies the checksum option for the associated file. This directs PAT to generate an octal value for the sum of all the binary data composing the module in that file (See Section 24.2.5 for more information on checksums.)
number	specifies an octal value. PAT compares the checksum value it computes for a module with the octal value you specify

23.2 How PAT Effects Updates

PAT updates a base input module by using additions and corrections you supply in a correction file. This section describes the PAT input and correction files, and gives information on how to create the correction file.

23.2.1 Input File

The input file is the file to be updated; it is the base for the output file and must be in object module format. When PAT executes, the module in the correction file applies to this file.

23.2.2 Correction File

The correction file must be in object module format and it is usually created from a MACRO–11 source file in the following format:

```
.TITLE inputname
[.IDENT updatenum]
[section name]
inputline
inputline
*
*
*
```

where:

inputname	is the name of the module to be corrected by the PAT update. That is, *inputname* must be the same name as the name on the input file .TITLE directive for a single module in the input file
updatenum	is any value acceptable to the MACRO–11 assembler. Generally, this value reflects the update version of the file being processed by PAT, as shown in the examples below
section name	is the ASECT, CSECT, or PSECT included in the correction file
inputline	are lines of input for PAT's use in correcting and updating the input file

During execution, PAT adds any new global symbols that are defined in the correction file to the module's symbol table. Duplicate global symbols in the correction file supersede their counterparts in the input file, provided that both definitions are relocatable or both are absolute.

A duplicate PSECT or CSECT supersedes the previous PSECT or CSECT, provided:

- both have the same relocatability attribute (ABS or REL)
- both are defined with the same directive (.PSECT or .CSECT)

If PAT encounters duplicate PSECT names, it sets the length attribute for the PSECT to the length of the longer PSECT and appends a new PSECT to the module.

If you specify a transfer address, it supersedes that of the module you are patching.

23.3 Updating Object Modules

The following examples show the source code for an input file and a correction file to be processed by PAT and the linker. The examples show as output a single source file that, if assembled and linked, would produce a binary module equivalent to the file generated by PAT and LINK. Two techniques are described: one is for overlaying lines in a module, and the other is for appending a subroutine to a module.

23.3.1 Overlaying Lines in a Module

In the following example, PAT first appends the correction file to the input file. The linker is then executed to replace code within the input file.

The input file for this example is:

```
        .TITLE  ABC
        .IDENT  /01/
        .ENABL  GBL
ABC::
        MOV     A,C
        JSR     PC,XYZ
        RTS     PC
        .END
```

To add the instruction ADD A,B after the JSR instruction, the following patch source file is included:

```
        .TITLE  ABC
        .IDENT  /01.01/
        .ENABL  GBL
.=.+12
        ADD     A,B
        RTS     PC
        .END
```

The patch source is assembled using MACRO–11 and the resulting object file is input to PAT along with the original object file. The following source code represents the result of PAT processing:

```
        .TITLE  ABC
        .IDENT  /01.01/
        .ENABL  GBL
ABC::
        MOV     A,C
        JSR     PC,XYZ
        RTS     PC
.=ABC
.=.+12
        ADD     A,B
        RTS.    PC
        .END
```

After the linker processes these files, the load image appears, as this source code representation shows:

```
        .TITLE  ABC
        .IDENT  /01.01/
        .ENABL  GBL
ABC::
        MOV     A,C
        JSR     PC,XYZ
        ADD     A,B
        RTS     PC
        .END
```

The linker uses the .=.+12 in the program counter field to determine where to begin overlaying instructions in the program and, finally, overlays the RTS instruction with the patch code:

```
ADD     A,B
RTS     PC
```

23.3.2 Adding a Subroutine to a Module

In many cases, a patch requires that more than a few lines be added to patch the file. A convenient technique for adding new code is to append it to the end of the module in the form of a subroutine. This way, you can insert a JSR instruction to the subroutine at an appropriate location. The JSR directs the program to branch to the new code, execute that code, and then return to in-line processing.

The source code for the input file for the example is:

```
        .TITLE  ABC
        .IDENT  /01/
        .ENABL  GBL
ABC::
        MOV     A,B
        JSR     PC,XYZ
        MOV     C,R0
        RTS     PC
        .END
```

Suppose you wish to add the instructions:

```
MOV     D,R0
ASL     R0
```

between

```
MOV     A,B
```

and

```
JSR     PC,XYZ
```

The correction file to accomplish this is as follows:

```
        .TITLE  ABC
        .IDENT  /01.01/
        .ENABL  GBL
        JSR     PC,PATCH
        NOP
        .PSECT  PATCH
PATCH:
        MOV     A,B
        MOV     D,R0
        ASL     R0
        RTS     PC
        .END
```

PAT appends the correction file to the input file, and the linker then processes the file, generating the following output file:

```
        .TITLE  ABC
        .IDENT  /01.01/
        .ENABL  GBL
ABC::
        JSR     PC,PATCH
        NOP
        JSR     PC,XYZ
        MOV     C,R0
        RTS     PC
        .PSECT  PATCH
PATCH:
        MOV     A,B
        MOV     D,R0
        ASL     R0
        RTS     PC
        .END
```

In this example, the JSR PC,PATCH and NOP instructions overlay the three-word MOV A,B instruction. (The NOP is included because this is a case where a two-word instruction replaces a three-word instruction. NOP is required to maintain alignment.) The linker allocates additional storage for .PSECT PATCH, writes the specified code into this program section, and binds the JSR instruction to the first address in this section. Note that the MOV A,B instruction, replaced by the JSR PC,PATCH, is the first instruction the PATCH subroutine executes.

23.4 Determining and Validating the Contents of a File

Use the checksum option (/C) to determine or validate the contents of a module. The checksum option directs PAT to compute the sum of all binary data composing a file. If you specify the command in the form /C:n, /C directs PAT to compute the checksum and compare that checksum to the value you specify as *n*.

To determine the checksum of a file, enter the PAT command line with the /C option applied to the appropriate file (the file whose checksum you want to determine). For example, PAT responds to the command

```
=INFILE/C,INFILE.PAT
```

with the message

```
?PAT-W-Input module checksum is nnnnnn
```

PAT generates a similar message when you request the checksum for the correction file.

To validate the changes made to a file, enter the checksum option in the form /C:n. PAT compares the value it computes for the checksum with the value you specify as *n*. If the two values do not match, PAT enters the changes but displays a message reporting the checksum error as either:

```
?PAT-W-Input file checksum error
```

or

```
?PAT-W-Correction file checksum error
```

Checksum processing always results in a nonzero value.

Do not confuse this checksum with the record checksum byte.

Chapter 24
Source Language Patch Program (SLP)

The Source Language Patch Program (SLP), is a patching tool you can use for maintaining source files that exist on any RT–11 device.

SLP accepts as input a source file you wish to patch and a command file that you create when you compare two source programs using the source compare program, SRCCOM, described in Chapter 15. When you use SLP along with the SRCCOM command file, you can quickly and easily patch one version of a source program to match another version.

Chapter 15, Source Compare Program (SRCCOM), describes the procedure you can use to create a patch command file that is suitable for input to SLP.

24.1 Calling and Using SLP

To call SLP from the system device, type the following in response to the keyboard monitor dot (.):

```
R SLP<RET>
```

The Command String Interpreter prints an asterisk (*) at the left margin of the terminal and waits for a command string. If you enter only a carriage return in response to the asterisk, SLP prints its current version number. You can type CTRL/C to halt SLP and return control to the monitor when SLP is waiting for input from the console terminal. To restart SLP, type R SLP or REENTER in response to the monitor's dot. Chapter 6, Command String Interpreter, describes the general syntax of the command line that SLP accepts.

Enter a command line according to this general syntax:

[outfil][,listfil]=infil,comfil/[option...]

where:

outfil — represents the updated source file. The default file type is .MAC

listfil — represents the listing file. When you specify this file, SLP creates a numbered listing of the updates SLP made to the source file. The default file type is .LST

infil — represents the source file you want SLP to update. The default file type is .MAC

comfil represents the command file that contains the commands for updating the source file. The default file type is .DIF. You can create this file by using SRCCOM with the /P option

/option represents one of the options listed in Table 24–1

Although either output files can be omitted, you must use one or both.

24.2 Options

Table 24–1 lists the options you can use in the command line.

Table 24–1: SLP Options

Option	Function
/A	Disables audit trail generation. The audit trail is a string of characters that SLP appends to the end of each updated line in the output files. The audit trail keeps track of the update status of each line in the output file. You can use the /A option if you do not want SLP to use the audit trail in both the updated source file and the listing file.
/B	Inserts spaces instead of tabs between the source line and the audit trail.
/D	Creates a double-spaced listing. When you use this option, SLP double-spaces between the lines in a listing file.
/L:n	Specifies the size of the source line, where *n* represents the maximum number of characters you want in the source line. The default buffer size for formatting lines is 200 (decimal) bytes. If you expect the size of the command lines or source lines to be greater than what can fit in the line buffer, you can use this option to change the buffer size. SLP interprets the number you specify for *n* as an octal number; if you enter a decimal number, use a decimal point. The line buffer must be at least as long as the sum of the column number where the audit trail begins and the number of characters in the audit trail.
/P:n	Specifies the start column of the audit trail, where *n* represents the column number in which you want the audit trail to start. If the number you specify for *n* is decimal, be sure to use a decimal point after the number. By default, SLP starts the audit trail in column 73 (decimal). If a source line extends beyond the column where the audit trail begins, the audit trail can overstrike the source line. If you use the /P:n option, you start the audit trail in any tab stop column. SLP rounds up the number you specify to the nearest tab stop column. If, for example, you specify 46 for n, SLP rounds this number to 49.
/S:n	Specifies size of the audit trail, where *n* represents the number of characters you want in the audit trail. If the number you specify is decimal, be sure to use a decimal point after the number. The default number of characters in the audit trail is 12 (decimal). The maximum number of characters you can specify for the audit trail is 16 (decimal).
/T	Retains trailing blanks and tabs in the input source file. By default, SLP removes spaces and tabs that appear at the end of lines in the input source file.

24.3 Example

This section uses SLP to patch the source file, ANTONY.MAC, so that it matches the source file CAESAR.MAC. CAESAR.MAC consists of the following lines.

```
FRIENDS, ROMANS, COUNTRYMEN!
LEND ME YOUR EARS!
I COME TO BURY CAESAR,
NOT TO PRAISE HIM.
THE EVIL THAT MEN DO
LIVES AFTER THEM.
THE GOOD IS OFT INTERRED
WITH THEIR BONES;
SO LET IT BE WITH CAESAR!
```

The file this example will patch, ANTONY.MAC, follows.

```
FRIENDS, ROMANS, COUNTRYMEN!
LEN ME YOUR EARS!
I COME TO BURY CAESAR,
NOT TO PRAISE HIM.
THE EVIL THAT MAN DO
LIVES AFTER THEM.
THE GOOD IS OFT ENTERED
WIT THEIR HOMES;
SO LET IT BE WITH CAESAR!
```

Using the /P option in SRCCOM, this example obtains a command file, CAESAR.DIF. CAESAR.DIF contains the necessary commands to make ANTONY.MAC match CAESAR.MAC. The following command line directs SLP to patch ANTONY.MAC so that it matches CAESAR.MAC.

```
.R SLP
*ANTONY,ANTONY=ANTONY,CAESAR
```

After executing the command above, SLP assigns a .BAK file type to the input file ANTONY.MAC. It assigns .MAC file type to the updated source file. SLP has also created a listing of ANTONY.MAC that lists each line by number and appends an audit trail to each new line. The updated file, ANTONY.MAC, is now identical to CAESAR.MAC. ANTONY.LST appears below.

```
SLP V04.00                          ANTONY,ANTONY=ANTONY.MAC,CAESAR,DIF

     1.  FRIENDS, ROMANS, COUNTRYMEN!          ;**NEW**
     2.  LEND ME YOUR EARS!                    ;**-1
     3.  I COME TO BURY CAESAR,
     4.  NOT TO PRAISE HIM.                    ;**NEW**
     5.  THE EVIL THAT MEN DO                  ;**-1
     6.  LIVES AFTER THEM.                     ;**NEW**
     7.  THE GOOD IS OFT INTERRED              ;**NEW**
     8.  WITH THEIR BONES;                     ;**-2
     9.  SO LET IT BE WITH CAESAR!
```

Note that when SLP updates a line, it appends an additional audit trail below the audit trail of the updated line. The additional audit trail keeps

track of the number of consecutive lines that have been updated. In ANTONY.LST, above, note the audit trails ;**–1 and ;**–2.

24.4 Creating and Maintaining a Command File

SLP is a line-oriented patching tool. That is, you make changes to entire lines, and not to individual or strings of characters within a line. If you want to change only a few characters within a line, it will be necessary for you to enter a new line.

Although DIGITAL recommends you use the SRCCOM /P option to create the SLP input command file, you can use any RT–11 editor to create it yourself. The section that follows describes the commands, or operators, you use to create the command file. This procedure is tedious, however, and in most cases unnecessary. But for completeness, this procedure is included with this chapter. Table 24–2 lists the commands, or operators, you enter into the command file.

The section ends with a description of various line manipulations that SLP can effect.

Table 24–2: SLP Command File Operators

Operator	Explanation
-	Indicates the start of an update.
\	Disables the audit trail. Note that this operator must appear on a line by itself.
%	Enables the audit trail.
/	Indicates the end of an update or a series of updates; it appears as the last character in the command file.
<	Serves as an escape character for characters SLP would otherwise interpret as operators. For example, if you want to include a slash (/) in a source file, type the less-than character (<) before the slash. Then, SLP will not interpret the slash as an operator. You can use the less-than character as an escape character for all SLP command file operators.

24.4.1 Update Line Format

The general format of the SLP command file update line follows.

```
-locator1,[locator2],[/audit trail/][;]
inputline
   .
   .
   .
```

where:

- indicates that this is an update line

locator1	represents a character string that serves as a line locator. SLP moves the line pointer to the line specified by the line locator. You can specify this line locator with any of the locator forms described below
locator2	represents a character string that, when used with *locator 1,* defines a range of lines you want to delete or replace. You can specify this line locator with any of the locator forms described below
/audit trail/	represents a character string you use as an audit trail. SLP appends the audit trail to the right of each updated line. You must delimit the audit trail with slashes (/)
inputline	represents a line of new text that SLP inserts into the file immediately following the current line. You can enter as many input lines as you want
;	is an optional command line terminator

All fields in the update line are positional. That is, if you specify only locator1 and an audit trail, you must use two commas between those two fields. If you want to specify only the audit trail, you must precede the audit trail with two commas.

The line locators can take one of the following forms:

```
/string/[+n]
/string...string/[+n]
number[+n]
.+n
```

where:

/string/[+n]	represents an ASCII character string. You must delimit any string you enter with slashes. SLP locates the first occurrence of this string, and moves the line pointer to the line that contains that string. *+n* represents the offset from the line that contains the string. You must use the plus character (+) with the n notation
/string...string/[+n]	represents an ASCII character string. SLP locates the line in which the two strings delimit a larger string. Use the ellipsis (...) in this locator form to separate the two strings. *+n* represents the offset from the line specified by the string...string locator
number[+n]	represents the line number to which SLP is to move the line pointer. *+n* represents the offset from the line specified by number

.+n	represents the offset from the current line pointer. SLP interprets the period (.) as the current line pointer location, and the *+n* as the offset from it. You must use the plus character (+)

24.4.2 Creating a Numbered Listing

You can use SLP to create a numbered listing of the input source file. In creating a command file, you should use a numbered listing when you prepare command input. To generate a numbered listing, enter the following lines:

```
R SLP
*,listfile=infile
```

Listfile represents the listing file SLP produces, and *infile* represents the input source file. Here is a file, PROG.MAC, from which SLP is to create a numbered listing:

```
        .TITLE  PROG.MAC   VERSION 1
        .MCALL  .TTYOUT, .EXIT, .PRINT

EXP:    .PRINT  #MESSAG
        MOV     #N,R5
FIRST:  MOV     #N+1,R0
        MOV     #A,R1
```

The following command line creates a numbered listing, PROG.LST of the file above, PROG.MAC:

```
*,PROG=PROG
```

After SLP processes the command above, it produces the following listing of PROG.MAC:

```
SLP  V04.00                          ,PROG=PROG.MAC

     1.             .TITLE  PROG.MAC   VERSION 1
     2.
     3.             .MCALL  .TTYOUT, .EXIT, .PRINT
     4.
     5.   EXP:      .PRINT  #MESSAG
     6.             MOV     #N,R5
     7.   FIRST:    MOV     #N+1,R0
     8.             MOV     #A,R1
```

24.4.3 Adding Lines to a File

To add lines to a file, enter in the command file one of the three locator forms below:

```
-number
-,[+n],,[/audittrail/]
-/string/
```

Notice in the second locator form the two commas between the locator and the audit trail. You do not have to insert these commas if you are not specifying an audit trail.

Below is a file, NUMBER.PAS, to which SLP is to add new lines.

```
PROGRAM NUMBER;

TYPE    TEXT    =FILE OF CHAR;
        PTR     =^WORDNODE;
        WORDNODE=RECORD
                  WORD:ARRAY[1..30] OF CHAR;
                  NEXT:PTR;
                  END;

VAR     P,TOP   :PTR;
        INTEXT  :TEXT;
        I       :INTEGER;
```

SLP is to insert the following line between the fourth and fifth lines of NUMBER.PAS.

The command file, OMEGA.DIF, contains the following update 1 to perform this procedure.

```
-/PTR/
(*POINTER TO NODE*)
/
```

When SLP processes OMEGA.DIF with NUMBER.PAS, it produces the following updated listing file.

```
SLP V04.00                        NUMBER.PAS,NUMBER=NUMBER.PAS,OMEGA.DIF

    1.  PROGRAM NUMBER;
    2.
    3.  TYPE    TEXT    =FILE OF CHAR;
    4.          PTR     =^WORDNODE;
    5.  (*POINTER TO NODE*)                                    ;**NEW**
    6.          WORDNODE=RECORD
    7.                    WORD:ARRAY[1..30] OF CHAR;
    8.                    NEXT:PTR;
    9.                    END;
   10.
   11.  VAR     P,TOP   :PTR;
   12.          INTEXT  :TEXT;
   13.          I       :INTEGER;
```

SLP has numbered the lines, inserted the new text, and appended the default audit trail (;**NEW**) to the new line.

The next example uses the same source file, but uses this command in the command file, SIGMA.DIF:

```
-/WORDNODE/+2
              ID  :INTEGER;
/
```

When SLP processes SIGMA.DIF with the source file NUMBER.PAS, it generates the following listing file:

```
SLP V04.00                          NUMBER.PAS,NUMBER=NUMBER.PAS,SIGMA.DIF

    1.  PROGRAM NUMBER;
    2.
    3.  TYPE    TEXT    =FILE OF CHAR;
    4.          PTR     =^WORDNODE;
    5.          WORDNODE=RECORD
    6.                    WORD:ARRAY[1..30] OF CHAR;
    7.                    NEXT:PTR;
    8.                    ID  :INTEGER;
    9.                    END;                          ;**NEW**
   10.
   11.  VAR     P,TOP   :PTR;
   12.          INTEXT  :TEXT;
   13.          I       :INTEGER;
```

Again, SLP has numbered the lines, and this time it skips two lines after the first occurrence of string WORDNODE before inserting the new input line.

24.4.4 Deleting Lines in a File

The SLP command file command for deleting lines from a file is:

-locator1,locator2,[/audittrail/][;]

where *locator1* and *locator2* can be any of the forms of the locator fields described earlier. *locator1* specifies the line where SLP is to begin deleting lines. *locator2* specifies the last line SLP is to delete.

If you want to delete lines five through eight in file NUMBER.PAS, it will be helpful to look at a numbered listing of NUMBER.PAS.

```
SLP V04.00                                        ,NUMBER=NUMBER.PAS

    1.  PROGRAM NUMBER;
    2.
    3.  TYPE    TEXT    =FILE OF CHAR;
    4.          PTR     =^WORDNODE;
    5.          WORDNODE=RECORD
    6.                    WORD:ARRAY[1..30] OF CHAR;
    7.                    NEXT:PTR;
    8.                    END;
    9.
   10.  VAR     P,TOP   :PTR;
   11.          INTEXT  :TEXT;
   12.          I       :INTEGER;
```

In the command file, GAMMA.DIF, the command for deleting lines five through eight follows.

```
-/WORDNODE/,/END/
/
```

When SLP processes GAMMA.DIF with NUMBER.PAS, it produces this listing file of NUMBER.PAS.

```
SLP V04.00                          NUMBER.PAS,NUMBER=NUMBER.PAS,GAMMA.DIF

     1.   PROGRAM NUMBER;
     2.
     3.   TYPE     TEXT     =FILE OF CHAR;
     4.                                                          ;**-4
     5.   VAR      P,TOP    :PTR;
     6.            INTEXT   :TEXT;
     7.            I        :INTEGER;
```

24.4.5 Replacing Lines in a File

When you replace lines, you delete and then add new text. To replace lines in a file, first enter the full SLP edit command for the delete operation. The first line locator specifies the first line to be deleted. The second line locator specifies both the last line to be deleted and the location where SLP is to insert new text. For example, the command file command instructs SLP to move the line pointer to line 4.

```
-4,.+4
```

Then, SLP is to delete the next four lines (represented by +4), including line 4. Finally, SLP is to insert input lines that follow in the command file. SLP inserts the new lines, beginning at the line pointer's current location.

The following example illustrates replacing lines in a file. The source file, BETA.MAC, consists of the following lines:

```
        .TITLE   BETA.MAC
        .MCALL   .TTYOUT, .PRINT, .EXIT
START:  .PRINT   #MESSAG
        MOV      #5,R0
```

The command file, DELTA.DIF, contains:

```
-5,+1,/AUDIT TRAIL/
BNE          START:
MOVB (R2),-(R3)
/
```

When SLP processes DELTA.DIF with BETA.MAC, it produces the following listing file:

```
SLP  V04.00  BETA,BETA=BETA,DELTA

1.           .TITLE   BETA.MAC
2.
3.           .MCALL   .TTYOUT, .PRINT, .EXIT
4.
5.   START:  .PRINT   #MESSAG
6.           BNE      START:                    ;AUDIT TRAIL
7.           MOVB      (R2),-(R3)               ;AUDIT TRAIL
8.           MOV      #5,R0
```

Chapter 25
Patch Utility (PATCH)

You can use the PATCH to make code modifications to any RT–11 file (see Table 3–2 in Chapter 3 for a complete list of RT–11 file types). You use PATCH to interrogate and then to change words or bytes in the file.

It is always a good idea to create a backup version of the file you want to patch, because PATCH makes changes directly to the file as you work.

25.1 Calling and Using PATCH

To call PATCH from the system device, respond to the dot (.) printed by the keyboard monitor by typing:

```
R PATCH <RET>
```

PATCH then prints:

```
FILE NAME --
*
```

You should enter the name of the file you want to patch according to this general syntax:

filespec[/option...]

where:

filespec represents the device, file name, and file type of the file you want to patch

/option is one of the options listed in Table 25–1

If you do not specify a file type, PATCH assumes a .SAV file type.

25.2 Options

Table 25–1 summarizes the options that are valid for PATCH at this point in the opening command.

Table 25–1: PATCH Options

Option	Meaning
/A	Use with a device specification with or without a file specification. Use without a file specification to repair damaged RT–11 directories on directory-structured devices or to patch the bootstrap on disk block 0. Use with a file specification when the file is a source file or has a file type other than .SAV. Use if the file is an RT–11 monitor file.
/O	Use if the file is an overlay-structured file.
/C	Requires you to enter a checksum. If you make no modifications, PATCH ignores the /C option.
/D	Use if you do not know the checksum for a particular patch. PATCH prints the checksum for that patch. If you make no modifications, PATCH ignores the /D option.

Note that you must enter the complete file specification and accompanying options at this point; they are not legal at any other time. If you enter a carriage return instead of a file specification, however, PATCH prints its current running version number. It then repeats the prompt for a file specification.

After you enter the file specification, PATCH prints another asterisk and waits for commands.

25.2.1 Checksum

The checksum option (/C) helps you verify your work. It lets you compare the patch that you make to another patch that is known to be correct. The checksum does not tell you specifically where your error is, but it does tell you that an inconsistency exists.

PATCH can maintain a running total of the value of each command, argument, and character you enter. This total is called the checksum for the patch.

For example, if you receive from DIGITAL a patch to improve your system's performance, the patch contains a checksum value. You should use the /C option in the first PATCH command line, and then make the modifications to your file exactly as shown in the DIGITAL patch. When you exit, PATCH asks you for a checksum. Enter the value from the DIGITAL patch. If the checksum you enter and the checksum that PATCH generated when you made your modifications do not match, PATCH prints the *?PATCH-W-Checksum error* message. You then know that you made an error in patching your file and that you need to try again.

25.3 Commands

Table 25–2 summarizes the PATCH commands. Upper-case characters represent PATCH commands; lower-case characters represent octal values or

ASCII characters. The sections that follow the table describe the commands in detail. Section 25.4 provides examples of PATCH.

Table 25–2: PATCH Commands

Command	Section	Explanation
v;nR	25.3.8	Sets relocation register *n* to value *v*.
x;B	25.3.7	Sets the bottom address of the overlay file to the value *x*.
r,o/	25.3.3	Opens the word location indicated by the contents of relocation register *r* plus offset *o*.
r,o\	25.3.3	Opens the byte location indicated by the contents of relocation register *r* plus offset *o*.
s:r,o/	25.3.3	Opens the word location indicated by the contents of relocation register *r* plus offset *o* in overlay segment *s*.
s:r,o\	25.3.3	Opens the byte location indicated by the contents of relocation register *r* plus offset *o* in overlay segment *s*.
RET	25.3.3	Closes the currently open word or byte.
LF	25.3.3	Closes the currently open word or byte and opens the next sequential word or byte.
^	25.3.3	Closes the currently open word or byte and opens the previous word or byte.
@	25.3.3	Closes the currently open word and opens the word it addresses.
F	25.3.1	Closes the file currently open and requests a new file specification.
E	25.3.2	Closes the file currently open and returns control to the system monitor.
x;O	25.3.5	Indicates that a value in the overlay handler or its tables is being modified to the value *x* and that the overlay structure must be reinitialized. A value of zero (0) is illegal and generates an error message.
&	25.3.6	Indicates that PATCH should add the contents of all subsequently opened locations to the checksum, until it encounters another & symbol.
A	25.3.4	Prints the contents of the opened word or byte as ASCII characters. (If a byte is open, one character prints; if a word is open, two characters print.)
X	25.3.4	Prints the contents of the opened word as an unpacked Radix–50 word.
C(x[x])	25.3.4	Resets the contents of the opened word or byte to the ASCII value you type. (If a byte is open, you must type one character; if a word is open, you must type two characters.)
P(xxx)	25.3.4	Resets the contents of the currently opened word to the packed Radix–50 value of the three ASCII characters you type (you must type three characters).

25.3.1 Patching a New File (F)

The F command causes PATCH to request you to enter a checksum, or it prints the required checksum (depending upon the options you specify). It also causes PATCH to close the currently open file and to print an asterisk indicating its readiness to accept another command string. No checksum dialogue is invoked if you have not previously specified checksum options (with /D or /C).

25.3.2 Exiting from PATCH (E)

The E command causes PATCH to close the currently open file and return control to the RT–11 monitor. The checksum dialogue is printed before the exit only if you specify the checksum options /D or /C.

25.3.3 Examining and Changing Locations in the File

For a non-overlaid file, you can open a word address (as with ODT) by typing:

[relocation register,]offset/

PATCH types the contents of the location and waits for you to enter either new location contents or another command.

For an overlay file, the format is:

[segment number:][relocation register,]offset/

Segment number represents the overlay segment number as it is printed on the link map for the file. If you omit the segment number, PATCH assumes the root segment. If you make an error in a command string while patching an overlaid program, you can use CTRL/U to cancel the command. However, PATCH assumes the entire line is incorrect and preserves only the previously set relocation registers. PATCH preserves the segment number only across the ^ and LF commands.

Similarly, you can open a byte address in a file. The format for non-overlaid files is:

[relocation register,]offset\

The format for overlay files is:

[segment number:][relocation register,]offset\

Once a location has been opened, you can optionally type in the new contents in the format:

[relocation register,]octal value

Follow this line with one of the control characters from Table 25–3.

Table 25–3: PATCH Control Characters

Character	Function
<RET>	Closes the current location by changing its contents to the new contents (if any), and awaits additional control input.
<LF>	Closes the current location by changing its content to the new contents (if any), and opens the next sequential word or byte.
^	Closes the current location by changing its contents to the new contents (if any), and opens the previous word or byte.
@	Closes the current word location, and opens the word it addresses (in the same segment if it is an overlay file).

25.3.4 Translating and Indirectly Modifying Locations with a File

After opening a location within a file, you can translate the contents into ASCII characters or into the equivalent of a Radix–50 packed word.

To obtain the ASCII equivalent of the opened location, type A after PATCH prints the contents in octal.

PATCH then translates the word or byte into two (or one, if a byte is opened) ASCII characters. In this example, a byte is opened:

```
*1,100\ 102     A = B <LF>
```

PATCH prints only the printable ASCII characters in the opened word or byte (all nonprinting characters, such as ASCII codes 0–37, are represented by the *?* character). In this example, a word is opened:

```
*1, 100/ 302     A = B?  <LF>
```

In the next example, a word is opened, and both ASCII characters are printable:

```
*1, 100/ 33502     A = B7 <RET>
```

In these examples, one or both of the characters cannot be printed:

```
*0, 400/ 466     A = 6?  <LF>

*1, 202/ 55001     A = ?Z <LF>

*616/ 401     A = ??  <RET>
```

To unpack a Radix–50 word as three ASCII characters, type X after PATCH prints the contents of the opened word.

PATCH then unpacks the opened word and prints three ASCII characters.

Note that you must open a word and not a byte.

If the word you open contains an illegal Radix–50 word, PATCH prints *???*. If the translated character is not printable, PATCH prints *?* in place of it.

Neither the A command nor the X command alters the contents of the open location; however, PATCH updates the checksum to reflect the fact that you have entered a new command.

You can specify the A and X commands in any order on the same command line without altering the contents of the open location. For example,

```
*1, 15022/     50553  X = MAC A = kQ
```

After examining the location, you can change the location. For example,

```
*45660/10146  A = F?  X = BX8 12122 <RET> or <LF>
```

If the same location is reopened, the following change appears:

```
*45660/12122
```

You can change the contents of a location to the ASCII code of the value you specify by using the C command. You can use the P command to change a word to the packed Radix–50 word of the three characters you specify. This example changes an open byte to the ASCII code for the letter Z:

```
*1, 115\ 101     C (Z) <RET>
```

Note that PATCH prints the parentheses; you type only the character Z.

When reopened, the byte contains the ASCII code for Z:

```
*1, 115\ 132
```

Similarly, PATCH inserts the ASCII code for two ASCII characters into the low-order and high-order bytes of one word. This example changes an open word to the ASCII code for AZ:

```
*0,10116/ 103523  C (AZ) ^
```

If reopened, the location contains the ASCII code for AZ:

```
*0,10116/ 55101   A = AZ
```

You can examine the same location in more conventional ways, as this example shows:

```
*0,10116\ 101 <LF>
0,10117\ 132
```

Similarly, you can use the P command to change the contents of an open word to the Radix–50 packed word equivalent of the three ASCII characters you specify. This example changes the Radix–50 word equivalent of SAV to REL:

```
*2:1,400/ 73376  X = SAV  P (REL)<RET>
```

25.3.5 Setting Values in the Overlay Handler Tables of a Program

Use the ;O command to effect any changes to the overlay handler tables in an overlaid program. For example,

```
*616/   1043    1100;O
*
```

This command line increases the size of the referenced overlay region by 35(8) words, or 58(10) bytes, to allow room for a patch. The value being modified is a value associated with the overlay handler tables, or a value required by the overlay handler for proper overlay structure initialization. The overlay structure is re-initialized and you can enter commands to modify the new region on the next line. A value of 0 is not permitted with the ;O command. If you omit the preceding argument, or use 0, an error message is printed on the terminal.

25.3.6 Including the Old Contents in the Checksum

Sometimes it is important that the present contents of the locations being changed have known specific values. This is the case when DIGITAL publishes system patches. The & command aids in implementing system patches. It automatically includes the old contents of an open location into the checksum. This command is a simple switch. The first occurrence of the & turns the switch on, the second turns it off. While the switch is on, the old contents of every location you open and close become part of the checksum. To use the & command, type:

```
&
```

PATCH then prints a carriage return-line feed sequence and another * indicating its readiness to accept another command. This switch is then enabled.

If you type the command on a line where a location is currently open, PATCH closes the location and resets the switch. PATCH then prompts with an asterisk indicating that it is ready to accept additional commands.

25.3.7 Setting the Bottom Address

To patch an overlay file, PATCH must know the bottom address at which the program was linked if it is different from the initial stack pointer. This is the case if the program sets location 42 in an .ASECT. To set the bottom address, type:

bottom address;B

You must issue the B command before you open any locations in an overlay for modification.

25.3.8 Setting Relocation Registers

You set the relocation registers 0–7 with the R command (as with ODT). The R command has the syntax:

relocation value;relocation registerR

Be careful when you type this command string. If you substitute a comma (,) for the semicolon (;) in the R command, PATCH does not generate an error message. However, it does not set the value you specify in the relocation register.

Once you set one of the eight relocation registers, the expression:

relocation register,octal number

in a command string will have the value:

relocation value + octal number

25.4 Examples

This section consists of two patch examples: one example for a nonoverlaid file, and the other for an overlaid file. In each case, the steps that were taken to assemble, link, and patch the files are illustrated.

25.4.1 Patching a Nonoverlaid File

The following command assembles the program PROMPT.MAC:

```
 MACRO/LIST PROMPT
ERRORS DETECTED:  0
```

The following listing was produced on the line printer as a result of the assembly. It consists of two parts: (1) the assembly listing of the source program, and (2) the symbol table listing.

```
PROMPT.MAC      MACRO V03.00 5-MAY-77 16:54:30 PAGE 1

      1                                                 .TITLE  PROMPT.MAC
      2                                                 .MCALL  .PRINT,.EXIT
      3                                                 .MCALL  .TTYOUT,.TTYIN
      4 000000                                          .CSECT  HGHSEG
      5                                                 .NLIST  BEX
      6 000000  112700  000052          START:  MOVB    #'*,R0          ; PRINT,...
      7 000004                                  .TTYOUT                 ; ...A PROMPT.
      8 000010                                  .TTYIN                  ; ACCEPT A CHARACTER FROM THE KEYBOARD
      9 000014  122700  000040                  CMPB    #' ,R0          ; IS IT A CONTROL CHARACTER?
     10 000020  101367                          BHI     START           ; YES - MUST BE A MISTAKE.
     11 000022  122700  000057                  CMPB    #'/,R0          ; NO - IS IT A "/"?
     12 000026  001011                          BNE     ERROR           ; NO - REPORT THE ERROR.
     13 000030                                  .TTYIN                  ; YES - GET NEXT.
     14 000034  122700  000126                  CMPB    #'V,R0          ; IS IT A "V" CHARACTER?
     15 000040  001004                          BNE     ERROR           ; NO - REPORT THE ERROR.
     16 000042                                  .PRINT  #MSG            ; YES - PRINT THE VERSION MESSAGE...
     17 000050                          EXIT:   .EXIT                   ; ...AND THEN EXIT TO THE RT-11 MONITOR.
     18 000052                          ERROR:  .PRINT  #CMDERR         ; PRINT THE "?(RET)(LF)"....
     19 000060  000747                          BR      START           ; ...AND THEN RESTART.
     20 000062     077      000         CMDERR: .ASCIZ  /?/
     21 000064     016      012     105 MSG:    .ASCIZ  <15><12>/FILE  V03.01
     22                                                 .LIST   BEX
     23        000000'                                  .END    START
```

(continued on next page)

```
PROMPT.MAC      MACRO V03.00 5-MAY-77 16:54:30 PAGE 1-1
SYMBOL TABLE

CMDERR  000062R     002 ERROR   000052R     002 EXIT    000050R     002 MSG     000064R     002 START  000000R     002

. ABS.  000000      000
        000000      001
HGHSEG  000107      002
ERRORS DETECTED:  0

VIRTUAL MEMORY USED:  562 WORDS ( 3 PAGES)
DYNAMIC MEMORY AVAILABLE FOR  58 PAGES
DK:PROMPT,LP:PROMPT=DK:PROMPT.MAC/C
```

The next command links file PROMPT.OBJ and produces an executable module called PROMPT.SAV.

```
.LINK/MAP  PROMPT
```

The following listing was produced on the line printer as a result of the link operation.

```
RT-11 LINK  V03.01       LOAD MAP          THU 05-MAY-77 16:55:28
PROMPT.SAV       TITLE:  PROMPT  IDENT:

SECTION  ADDR    SIZE    GLOBAL  VALUE    GLOBAL  VALUE    GLOBAL  VALUE

. ABS.  000000  001000    (RW,I,GBL,ABS,OVR)
HGHSEG  001000  000110    (RW,I,GBL,REL,OVR)

TRANSFER ADDRESS = 001000, HIGH LIMIT = 001110 =  292. WORDS
```

The program contains an error. On line 21 the characters *<16>* should be *<15>*. The following example uses PATCH to correct the error.

```
.R PATCH

FILE NAME--
*PROMPT/D
*1000;1R
*1;64\  16      15
*E

?PATCH-I-Checksum = 30633
.
```

The example shown above uses the /D option, which requests PATCH to print the checksum when the operation completes. Next, relocation register 1 is set to the transfer address, which the link map shows is 1000. The next command opens relative location 64, which contains the error, as the assembly listing shows at line sequence number 21. The value 15 is substituted for 16 (by typing 15 followed by a carriage return), and the exit command is issued (E). PATCH then prints the checksum for the operation, which is 30633.

The next example verifies the change just made.

```
.R PATCH

FILE NAME--
*PROMPT
*1000;1R
*1,64\   15
1,65\    12
1,66\    106
1,67\    111
1,70\    114
1,71\    105
1,72\    40
1,73\    40
1,74\    126
1,75\    60
1,76\    63
1,77\    56
1,100\   60
1,101\   61
1,102\   40
1,103\   40
1,104\   15
1,105\   12
*E
.
```

As before, relocation register 1 is set to 1000 and location 64 is opened. Now it contains the correct value, 15. The rest of the command lines are terminated with a line feed. This closes the open location and opens the next one. This example shows the values from line 21 of the assembly listing *<RET><LF> FILE V03.01 <RET><LF>* as they are stored in memory.

25.4.2 Patching an Overlaid File

The following commands assemble two programs: PTCH.MAC, the main program, and OVRLAY.MAC, the overlay.

```
.MACRO/LIST PTCH
ERRORS DETECTED:  0

.MACRO/LIST OVRLAY
ERRORS DETECTED:  0
.
```

The following listings were produced by the two assemblies.

```
PTCH.MAC       MACRO V03.00 5-MAY-77 17:58:35 PAGE 1

     1                                                  .TITLE  PTCH,MAC
     2                                                  .MCALL  .PRINT,.EXIT
     3 000000                                           .CSECT  HGHSEG
     4                                                  .GLOBL  ENTRY,MSG1
     5 000000  000403              START:  BR      EXIT          ; BRANCH IMMEDIATELY TO CALL OVERLAY,
     6 000002  012700  000016'             MOV     #MSG,R0       ; ALTERNATIVELY PRINT A MESSAGE
     7 000006                              .PRINT                ; DO THE PRINT,
     8 000010  004767  000000G     EXIT:   JSR     PC,ENTRY      ; CALL IN THE OVERLAY,
     9 000014                              .EXIT                 ; THEN EXIT ON RETURN,
    10                                                  .NLIST  BEX
    11 000016     015     012     124  MSG:    .ASCIZ  <15><12>/THIS IS A SUCCESSFUL PATCH/<15><12>
    12 000055     015     012     124  MSG1:   .ASCIZ  <15><12>/THIS IS AN OVERLAY PATCH/<15><12>
    13                                                  .LIST   BEX
    14         000000'                                  .END    START
```

(continued on next page)

```
PTCH.MAC        MACRO V03.00 5-MAY-77 17:58:35 PAGE 1-1
SYMBOL TABLE

ENTRY = ****** G        EXIT    000010R     002 MSG     000016R     002 MSG1    000055RG    002 START   000000R     002

. ABS.  000000     000
        000000     001
HGHSEG  000112     002
ERRORS DETECTED:  0

VIRTUAL MEMORY USED:  509 WORDS  ( 2 PAGES)
DYNAMIC MEMORY AVAILABLE FOR  59 PAGES
DK:PTCH,LP:PTCH=DK:PTCH


OVRLAY.MAC      MACRO V03.00 5-MAY-77 17:58:57 PAGE 1

     1                                                      .TITLE  OVRLAY.MAC
     2                                                      .MCALL  .PRINT
     3 000000                                               .CSECT  OVLSEG
     4                                                      .GLOBL  MSG1,ENTRY
     5 000000  000403                ENTRY:  BR      RETURN          ; BRANCH IMMEDIATELY TO RETURN.
     6 000002  012700  000000G               MOV     #MSG1,R0        ; ALTERNATIVELY PRINT A MESSAGE
     7 000006                                .PRINT
     8 000010  000207                RETURN: RTS     PC              ; THEN RETURN.
     9         000001                                        .END


OVRLAY.MAC      MACRO V03.00 5-MAY-77 17:58:57 PAGE 1-1
SYMBOL TABLE

ENTRY   000000RG    002 MSG1  = ****** G        RETURN  000010R     002

. ABS.  000000     000
        000000     001
OVLSEG  000012     002
ERRORS DETECTED:  0

VIRTUAL MEMORY USED:  354 WORDS  ( 2 PAGES)
DYNAMIC MEMORY AVAILABLE FOR  59 PAGES
DK:OVRLAY,LP:OVRLAY=DK:OVRLAY
```

The next command links the module PTCH.OBJ and the overlay module OVRLAY.OBJ, producing the executable module ROOT.SAV.

```
.LINK/MAP/PROMPT/EXECUTE:ROOT PTCH
*OVRLAY/O:1
*//

.
```

The following listing is the load map that resulted from this link.

```
RT-11 LINK  V03.01      LOAD MAP        THU 05-MAY-77 17:59:51
ROOT  .SAV      TITLE:  PTCH,M  IDENT:

SECTION  ADDR   SIZE    GLOBAL  VALUE   GLOBAL  VALUE   GLOBAL  VALUE

. ABS.  000000  001122    (RW,I,GBL,ABS,OVR)
HGHSEG  001122  000112    (RW,I,GBL,REL,OVR)
                        MSG1    001177
 SEGMENT SIZE = 001234 =   334. WORDS
 OVERLAY REGION 000001  SEGMENT 000001
OVLSEG  001236  000012    (RW,I,GBL,REL,OVR)
                        ENTRY @ 001236
 SEGMENT SIZE = 000012 =     5. WORDS


TRANSFER ADDRESS = 001122, HIGH LIMIT = 001250 =  340. WORDS
```

The following example shows how to patch an overlay segment.

```
.R PATCH

FILE NAME--
*ROOT/O/C
*1236;1R
*1:1,O/ 403      240
*E

Checksum?   45475

.R ROOT

THIS IS AN OVERLAY PATCH


.
```

The options /O and /C are used in the file specification. /O indicates that the file is overlaid. /C causes PATCH to verify that the changes are correct by requesting a checksum value that it compares to the actual checksum value the changes generate internally. The patch for this example was supplied by an experienced user, and the checksum for the correct patch is known to be 45475.

The first command line sets relocation register 1 to the start of the overlay segment to be patched. The load map shows that overlay segment 1 begins at location 1236. The next command opens the first location in overlay segment 1. It contains a branch instruction (403). A no-op instruction is substituted for it (240) followed by a carriage return, and E is used to exit. PATCH then requests the checksum value and 45475 is entered. This matches the checksum that the changes generated internally, so control returns to the monitor and the patch is successful.

The program is executed by typing:

```
R ROOT <RET>
```

Control branches immediately to the overlay segment. Because the branch instruction at ENTRY: is now inoperative, control passes to the next line and the message THIS IS AN OVERLAY PATCH prints on the terminal.

NOTE

The linker allocates space for overlay segments in 256-word blocks. Each segment begins on a block boundary. If a particular overlay segment's size is less than a whole number multiple of 256, you can add patches in the free space that exists between the end of that overlay segment and the beginning of the next block. You must first modify the word-count word in the overlay handler table so the patches you add are included in the size of the overlay segment. Be careful not to patch into the next block, though, because the next overlay segment begins there.

Appendix A
BATCH

RT–11 BATCH is a complete job control language that allows RT–11 to operate unattended. BATCH processing is ideally suited to frequently run production jobs, large and long-running programs, and programs that require little or no interaction with you, the user. With BATCH, you can prepare your job on any RT–11 input device and leave it for the operator to start and run.

RT–11 BATCH permits you to:

- Execute an RT–11 BATCH stream from any RT–11 input device
- Output a log file to any RT–11 output device (except magtape or cassette)
- Execute the BATCH stream with the single-job monitor or in the background with the foreground/background monitor or the extended memory monitor
- Generate and support system-independent BATCH language jobs
- Execute RT–11 monitor commands from the BATCH stream

RT–11 BATCH consists of: (1) the BATCH compiler, and (2) the BATCH run-time handler. The BATCH compiler reads the batch input stream you create, translates it into a format suitable for the RT–11 BATCH run-time handler, and stores it in a file. The BATCH run-time handler executes this file with the RT–11 monitor. As each command in the batch stream executes, BATCH lists the command, along with any terminal output generated, by executing the command on the BATCH log device.

A.1 Hardware and Software Requirements

You can run RT–11 BATCH on any single-job system that is configured with at least 12K words of memory. You need a minimum system of 16K words of memory to run BATCH in the background in a foreground/background environment. BATCH can run in any extended memory environment. A line printer, although optional, is highly desirable as the log device.

BATCH uses certain RT–11 system programs to perform its operations. For example, the $BASIC command executes the file BASIC.SAV. Make sure that the following RT–11 programs are on the system device, with exactly the following names, before you run BATCH:

BASIC.SAV (BASIC users only)
BA.SYS
BATCH.SAV

CREF.SAV (MACRO users only)
SYSLIB.OBJ (FORTRAN and MACRO users)
FORTRA.SAV (FORTRAN users only)
LINK.SAV
MACRO.SAV (MACRO users only)
PIP.SAV
DIR.SAV

A.2 Control Statement Format

For input to RT–11 BATCH, you can generate a file with the RT–11 editor and use any RT–11 input device, or you can use punched cards from the card reader. In both cases, the input consists of BATCH control statements. A BATCH control statement is divided into three fields, separated from one another by spaces: (1) command fields, (2) specification fields, and (3) comment fields. The control statement has the syntax:

$command/option specification/option [!comment]

Each control statement requires a specific combination of command and specification fields and options (see Section A.4). Control statements cannot be longer than 80 characters, excluding multiple spaces, tabs, and comments. You can use a hyphen (-) as a line continuation character to indicate that the control statement is continued on the next line (see Table A–4). Even if you use the line continuation character, the maximum control statement length is still 80 characters.

The following example of a $FORTRAN command illustrates the various fields in a control statement.

```
$FORTRAN/LIST/RUN  PROGA/LIBRARY  PROGB/EXE  !RUN FORTRAN
 command/options        spec fields/options       comment field
```

A.2.1 Command Fields

The command field in a BATCH control statement indicates the operation to be performed. It consists of a command name and certain command field options. Indicate the command field with a $ in the first character position and terminate it with a space, tab, blank, or carriage return.

A.2.1.1 Command Names — The command name must appear first in a BATCH control statement and have a dollar sign ($) in the first position of the command (for example, $JOB). No intervening spaces are allowed in the command name. BATCH recognizes only two forms of a command name: the full name, and an abbreviation consisting of $ and the first three characters of the command name. For example, you can enter the $FORTRAN command as:

```
$FORTRAN
```

or

```
$FOR
```

You cannot enter it as:

```
$FORT
```

or

```
$FORTR
```

A.2.1.2 Command Field Options — Options that appear in a command field are command qualifiers. Their functions apply to the entire control statement. All option names must begin with a slash (/) that immediately follows the command name. Table A–1 describes the command field options for BATCH and indicates the commands on which you can use them. Those option characters that appear in square brackets are optional. The command field options are described in greater detail in the sections dealing with the appropriate commands.

NOTE

All /NO options are the defaults, except the /WAIT option in the $MOUNT and $DISMOUNT commands and the /OBJECT option in the $LINK command.

Table A–1: Command Field Options

Option	Explanation
/BAN[NER]	Prints the header of the job on the log file. BATCH allows this option only on the $JOB command. Note that BATCH outputs the $JOB command line to the log device sixty times.
/NOBAN[NER]	Does not print a job header.
/CRE[F]	Produces a cross-reference listing during compilation. BATCH allows this option only on the $MACRO command.
/NOCRE[F]	Does not create a cross-reference listing.
/DEL[ETE]	Deletes input files after the operation completes. BATCH allows this option on the $COPY and $PRINT commands.
/NODEL[ETE]	Does not delete input files after operation completes.

/DOL[LARS]	The data following this command can have a $ in the first character position of a line. BATCH allows this option on the $CREATE, $DATA, $FORTRAN, and $MACRO commands. BATCH terminates reading data when you use one of the following commands or when it encounters a physical end-of-file on the BATCH input stream:
	`$JOB` `$EOD`
	`$SEQUENCE` `$EOJ`

/NODOL[LARS]	The data following this command cannot have a $ in the first character position; a $ in the first character position means a BATCH control command.

(continued on next page)

Table A–1: Command Field Options (Cont.)

Option	Explanation
/LIB[RARY]	Includes the default library in the link operation. BATCH allows this option on the $LINK and $MACRO commands.
/NOLIB[RARY]	Does not include the default library in the link operation.
/LIS[T]	Produces a temporary listing file (see Section A.2.5) on the listing device (LST:) or writes data images on the log device (LOG:). BATCH allows this option on the $BASIC, $CREATE, $DATA, $FORTRAN, $JOB, and $MACRO commands. When you use /LIST on the $JOB command, /LIST sends data lines in the job stream to the log device (LOG:).
/NOLIS[T]	Does not produce a temporary listing file.
/MAP	Produces a temporary link map on the listing device (LST:). BATCH allows this option on the $FORTRAN, $LINK, and $MACRO commands.
/NOMAP	Does not create a MAP file.
/OBJ[ECT]	Produces a temporary object file as output from compilation or assembly (see Section A.2.5). BATCH allows this option on the $FORTRAN, $LINK, and $MACRO commands. When you use /OBJECT on $LINK, BATCH includes temporary files in the link operation.
/NOOBJ[ECT]	Does not produce an object file as output of compilation; with $LINK, does not include temporary files in the link operation.
/RT11	Sets BATCH to operate in RT–11 mode (see Section A.5). BATCH allows this option only on the $JOB command.
/NORT11	Does not set BATCH to operate in RT–11 mode.
/RUN	Links (if necessary) and executes programs compiled since the last "link-and-go" operation or start of job. BATCH allows this option on the $BASIC, $FORTRAN, $LINK, and $MACRO commands.
/NORUN	Does not execute or link and execute the program after performing the specified command.
/TIM[E]	Writes the time of day to the log file when BATCH executes. BATCH allows this option only on the $JOB command. This command writes the time after each command that begins with a dollar sign ($).
/NOTIM[E]	Does not write the time of day to the log file.
/UNI[QUE]	Checks for unique spelling of options and keynames (see Section A.4.13). BATCH allows this option only on the $JOB command.
/NOUNI[QUE]	Does not check for unique spelling.
/WAI[T]	Pauses for operator action. BATCH allows this option on the $DISMOUNT, $MESSAGE, and $MOUNT commands.
/NOWAI[T]	Does not pause for operator action.
/WRI[TE]	Indicates that the operator is to WRITE-ENABLE a specified device or volume. BATCH allows this option only on the $MOUNT command.
/NOWRI[TE]	Indicates that no writes are allowed or that the specified volume is read-only; informs the operator, who must WRITE-LOCK the appropriate device.

A.2.2 Specification Fields

Specification fields immediately follow command fields in a BATCH control statement and apply only to the fields they follow. Use them to name the devices and files involved in the command. You must separate these fields from the command field, and from each other, by blanks or spaces.

If a specification field contains more than one file to be used in the same operation, separate the files by a plus (+) sign. For example, to assemble files F1 and F2 to produce an object file F3 and a temporary listing file, type:

```
$MACRO/LIST F1+F2/SOURCE F3/OBJECT
```

If you need to repeat a command for more than one field specification, separate the files by a comma (,). For example, the following command assembles F1 to produce F2, a temporary listing file, and a map file F3. It then assembles F4 and F5 to produce F6 and a temporary listing file.

```
$MACRO/LIST F1/SOURCE F2/OBJECT F3/MAP,F4+F5/SOURCE-
 F6/OBJECT
```

Depending on the command you use, specification fields can contain a device specification, file specification, or an arbitrary ASCII string. You can use an appropriate specification field option (see Table A–3) with any of these three items.

A.2.2.1 Physical Device Names — Represent each device in an RT–11 BATCH specification field with a standard two- or three-character device name. Table 3–1 in Chapter 3 lists each name and its related device. If you do not specify a unit number for devices that have more than one unit, BATCH assumes unit 0.

In addition to the permanent names shown in Table 3–1, you can assign logical device names to devices. A logical device name takes precedence over a physical name, thus providing device independence. With this feature, you do not need to rewrite a program that is coded to use a specific device if the device is unavailable. For example, DK: is initially assigned to the system device, but you can assign that name to diskette unit 1 (DX1:) with an RT–11 monitor ASSIGN command.

You must assign certain logical names prior to running any BATCH job. BATCH uses these logical names as default devices. These names are:

LOG: BATCH log device (cannot be magtape or cassette)
LST: Default for listing files generated by BATCH stream

The following are not legal device names in RT–11; if you use them, the operator must assign them as logical names with the ASSIGN command. You can use these names in BATCH streams written for other DIGITAL systems.

DF: Fixed-head disk (RF)

LL: Line printer with upper- and lower-case characters

M7: 7-track magtape

M9: 9-track magtape

PS: Public storage (DK: as assigned by RT–11)

Refer to Sections 4.3 and A.7.1 for instructions on assigning logical names to devices.

A.2.2.2 File Specifications — You can reference files symbolically in a BATCH control statement with a name of up to six alphanumeric characters followed, optionally, by a period and a file type of three alphanumeric characters. Tabs and embedded spaces are not allowed in either the file name or file type. The file type generally indicates the format of a file. It is good practice to conform to the standard file types for RT–11 BATCH. If you do not specify a file type for an output file, BATCH and most other RT–11 system programs assign appropriate default file types. If you do not specify a file type for an input file, the system searches for that file name with a default file type. Table A–2 lists the standard file types used in RT–11 BATCH.

Table A–2: BATCH File Types

File Type	Explanation
.BAS	BASIC source file (BASIC input)
.BAT	BATCH command file
.CTL	BATCH control file generated by the BATCH compiler
.CTT	BATCH temporary file generated by the BATCH compiler
.DAT	BASIC or FORTRAN data file
.DIR	Directory listing file
.FOR	FORTRAN IV source file (FORTRAN input)
.LST	Listing file
.LOG	BATCH log file
.MAC	MACRO source file (MACRO or SRCCOM input)
.MAP	Link map output from $LINK operation
.OBJ	Object file output from compilation or assembly
.SOU	Temporary source file
.SAV	Runnable file or program image output from $LINK

A.2.2.3 Wild Card Construction — You may use wild cards in certain BATCH control statements (such as, $COPY, $CREATE, $DELETE, $DIRECTORY, $PRINT). You can use the asterisk as a wild card to designate the entire file name or file type. See Chapter 4 for a complete description of the wild card construction.

NOTE

You cannot use embedded wild cards (* or %) in BATCH control statements. However, you can use them in the keyboard monitor commands if you use the RT–11 mode of BATCH.

A.2.2.4 Specification Field Options — Specification field options follow file specifications in a BATCH control statement and designate how the file will be used. These options apply only to the field in which they appear. Option names begin with a slash. The specification field options for RT–11 BATCH are listed in Table A–3. Optional characters in the option names are in square brackets.

Table A–3: Specification Field Options

Option	Explanation
/BAS[IC]	BASIC source file
/EXE[CUTABLE]	Indicates the executable program image file to be created as the result of a link operation
/FOR[TRAN]	FORTRAN source file
/INP[UT]	Input file; default if you specify no options
/LIB[RARY]	Library file to be included in link operation (prior to default library)
/LIS[T]	Listing file
/LOG[ICAL]	Indicates that the device is a logical device name; use in $DISMOUNT and $MOUNT commands
/MAC[RO]	MACRO source file
/MAP	Linker map file
/OBJ[ECT]	Object file (output of assembly or compilation)
/OUT[PUT]	Output file
/PHY[SICAL]	Indicates physical device name
/SOU[RCE]	Indicates source file
/VID	Volume identification

A.2.3 Comment Fields

Comment fields, which document a BATCH stream, are identified by an exclamation point (!) appearing anywhere except in the first character position of the control statement. BATCH treats any character following the ! and preceding the carriage return/line feed combination as a comment. For example:

```
$RUN PIP      !DELETE FILES ON DK:
```

This command runs the RT–11 system program PIP. BATCH ignores the comment.

You can also include comments as separate comment lines by typing a $ in character position 1, followed immediately by the ! operator and the comment. For example:

```
$!DELETE FILES ON DK:
```

A.2.4 BATCH Character Set

The RT–11 BATCH character set is limited to the 64 upper-case characters (ASCII 40 through 137). The current ASCII set is assumed (character 137 is underscore and not left arrow, and character 136 is circumflex, not up-arrow). The BATCH job control language does not support any control characters other than tab, carriage return, and line feed.

Table A–4 shows how BATCH normally interprets certain characters. Character interpretations are different if you use RT–11 mode (see Section A.5).

Table A–4: Character Explanation

Character	Explanation
space	Specification field delimiter. It separates arguments in control statements. BATCH considers any string of consecutive spaces and tabs (except in quoted strings) as a blank (that is, equivalent to a single space).
!	Comment delimiter. The input routine ignores all characters after the exclamation point, up to the carriage return/line feed combination.
"	Passes a text string containing delimiting characters where the normal precedence rules would create the wrong action. For example, use it to include a space in a volume identification (/VID).
$	BATCH control statement recognition character. A dollar sign ($) in the first character position of a BATCH input stream line indicates that the line is a control statement.
.	Delimiter for file type.
-	Indicates line continuation if the character after the hyphen is one of the following: • A carriage return/line feed • Any number of spaces or tabs followed by a carriage return/line feed • A comment delimiter (!) • Spaces followed by a comment delimiter (!) If any other character follows the hyphen, the hyphen is assumed to be a minus sign indicating a negative value in an option.
/	Precedes an option name. An alphanumeric string must immediately follow it.
0–9	Numeric string components.
:	Immediately follows a device name. You can also use it to separate an option name from its value or to separate an option value from its subvalue (you can use : interchangeably with = for this purpose).

(continued on next page)

Table A–4: Character Explanation (Cont.)

Character	Explanation
A-Z	Alphabetic string components.
=	Separates an option name from a value.
\	Illegal character except when it precedes a directive to the BATCH run-time handler from the operator (see Section A.7.3). (To include \ in an RT–11 mode command, use \\.)
+	File delimiter. Separates multiple files in a single specification field. Also indicates a positive value in options.
,	Separates sets of arguments for which the command is to be repeated.
*	A wild card in utility command file specifications.
CR/LF	Carriage return/line feed. It indicates end-of-line (or end of logical record) for records in the BATCH input stream.

A.2.5 Temporary Files

When you do not include field specifications in a BATCH command line, BATCH sometimes generates temporary files. For example, you can enter a $FORTRAN command that is followed in the BATCH stream by the FORTRAN source program as:

```
$FORTRAN/RUN/OBJECT/LIST
  FORTRAN source program
$EOD
```

This command generates: (1) a temporary source file from the source statements that follow, (2) a temporary object file, (3) a temporary listing file, and (4) a temporary memory image file.

BATCH sends temporary files to the default device (DK:) or the listing device (LST:) according to their type. If the device is file-structured, BATCH assigns file names and file types as follows:

nnnmmm.LST	for temporary listing files (sent to LST:)
nnnmmm.MAP	for temporary map files (sent to LST:)
nnnppp.OBJ	for temporary object files (sent to DK:)
000000.SAV	for temporary memory image files (sent to DK:)
nnnppp.SOU	for temporary source files (sent to DK:)

where:

nnn represents the last three digits of the sequence number assigned to the job by the $SEQUENCE command (see Section A.4.22). Thus, a sequence number of 12345 produces a file name beginning 345. If you do not use the $SEQUENCE command, BATCH sets *nnn* to 000

mmm represents the number of listing (or map) files BATCH generated since the BATCH run-time handler (BA.SYS) was loaded. The first such file, listing or map, is 000. Each time BATCH generates a new temporary file, it increments the file name by 1. Thus, the second listing file produced under job sequence number 12345 is 345001.LST, and the first map file produced is 345000.MAP

ppp represents the number of object or source files in the current BATCH run. The first such file (object or source) is 000. Each time BATCH generates a new temporary file, it increments the file name by 1. BATCH resets these file names to 000 every time you run BATCH and after every $LINK, $MACRO, or $FORTRAN command that uses the temporary files

A.3 General Rules and Conventions

You must adhere to the following general rules and conventions associated with RT–11 BATCH processing.

1. Always place a dollar sign ($) in the first character position of a command line.
2. Each job must have a $JOB and $EOJ command (or card).
3. You can spell out command and option names entirely or you can specify only the first three characters of the command and required characters of the option.
4. Specify wild card construction (*) only for the utility commands ($COPY, $CREATE, $DELETE, $DIRECTORY, and $PRINT) and for commands that normally accept wild cards in RT–11 mode.
5. Include comments at the end of command lines or in a separate comment line. When you include comments in a command line, place them after the command but precede them by an exclamation mark.
6. Include only 80 characters per control statement (card record), excluding multiple spaces, tabs, and comments.
7. When you omit file specifications from BATCH commands and supply data in the BATCH stream, the system creates a temporary file with a default name (see Section A.2.5).
8. You can use the RT–11 monitor type-ahead feature only with BATCH handler directives (see Section A.7.3) to be inserted into a BATCH program. No other terminal input (except input to a foreground program) can be entered while a BATCH stream is executing.
9. You cannot use an indirect command file to call BATCH.

A.4 Commands

Place BATCH commands in the input stream to indicate to the system which functions to perform in the job. All BATCH commands have a dollar sign ($) in the first character position (for example, $JOB). Intervening spaces are not allowed in command names. The command name must always start in the first character position of the line (card column 1).

BATCH commands are presented in alphabetical order in this chapter for ease of reference. However, if you are not familiar with BATCH, read the commands in a functional order as listed in Table A–5. The characters shown in square brackets are optional.

Table A–5: BATCH Commands

Command	Section	Explanation
$SEQ[UENCE]	A.4.22	Assigns an arbitrary identification number to a job.
$JOB	A.4.13	Indicates the start of a job.
$EOJ	A.4.11	Indicates the end of a job.
$MOU[NT]	A.4.18	Signals the operator to mount a volume on a device and optionally assigns a logical device name.
$DIS[MOUNT]	A.4.9	Signals the operator to dismount a volume from a device and deassigns a logical device name.
$FOR[TRAN]	A.4.12	Compiles a FORTRAN source program.
$BAS[IC]	A.4.1	Compiles a BASIC source program.
$MAC[RO]	A.4.16	Assembles a MACRO source program.
$LIB[RARY]	A.4.14	Specifies libraries for BATCH to use in link operations.
$LIN[K]	A.4.15	Links modules for execution.
$RUN	A.4.21	Causes a program to execute.
$CAL[L]	A.4.2	Transfers control to another BATCH file, executes that BATCH file, and returns to the calling BATCH stream.
$CHA[IN]	A.4.3	Passes control to another BATCH file.
$DAT[A]	A.4.6	Indicates the start of data.
$EOD	A.4.10	Indicates the end of data.
$MES[SAGE]	A.4.17	Issues a message to the operator.
$COP[Y]	A.4.4	Copies files.
$CRE[ATE]	A.4.5	Creates new files from data included in the BATCH stream.
$DEL[ETE]	A.4.7	Deletes files.
$DIR[ECTORY]	A.4.8	Provides a directory of the specified device.
$PRI[NT]	A.4.19	Prints files.
$RT[11]	A.4.20	Specifies that the following lines are RT–11 mode commands.

For each command listed below, the term filespec represents a device name, or file name, and a file type. Filespec has this form:

dev:filnam.typ

As a general rule, BATCH assumes device DK: if you omit a device specification.

A.4.1 $BASIC

The $BASIC command calls RT–11 single-user BASIC to execute a BASIC source program. The $BASIC command has the following syntax:

$BASIC[/option...] [filespec/option]] [!comments]

where:

/option — indicates an option you can append to the $BASIC command. The options are as follows:

- /RUN — Indicates that BATCH should execute the source program.
- /NORUN — Indicates that BATCH should only compile the program and send error messages to the log file.
- /LIST — Writes data images that are contained in the job stream to the log file (LOG:).
- /NOLIST — Writes data images to the log file only if you specify $JOB/LIST.

filespec — indicates the name and type of the source file and the device on which it resides. If you omit the file type, BATCH assumes .BAS. If you omit this specification, the source statements must immediately follow the $BASIC command in the input stream.

Terminate the source program after a $BASIC statement with either a $EOD command or with any other BATCH command that starts with a $ in the first position

/option — indicates an option that can follow the source file name. BATCH assumes any file name with no option appended is the name of a source file. This option can have one of the following values (or you can omit it):

- /BASIC — Indicates that the file name you specify is a BASIC source program.
- /SOURCE — Performs the same function as /BASIC.
- /INPUT — Performs the same function as /BASIC.

You can follow the $BASIC command with the source program, BASIC commands (such as RUN), or data. The following two BATCH streams, for example, produce the same results (but BATCH does not echo the same output format for both streams).

```
$BASIC                  $BASIC/RUN
10 INPUT A              10 INPUT A
20 PRINT A              20 PRINT A
30 END                  30 END
RUN                     $DATA
123                     123
$EOD                    $EOD
```

A.4.2 $CALL

The $CALL command transfers control to another BATCH control file, temporarily suspending execution of the current control file. BATCH executes the called file until it reaches $EOJ or until the job aborts; control then returns to the statement following the $CALL in the originating BATCH control file. You can nest calls up to 31 levels. BATCH includes the log file for the called file in the log file for the originating BATCH program. (See NOTE following the $EOJ command.)

The syntax of the $CALL command is :

$CALL filespec[!comments]

Options are not allowed in the $CALL command. BATCH saves $JOB command options across a $CALL; however, they do not apply to the called BATCH file. If you specify .CTL as the file type, BATCH assumes a precompiled BATCH control file. If you do not specify a file type, BATCH assumes .BAT and compiles the called BATCH stream before execution.

NOTE

If the called program generates temporary files, those files can supersede existing temporary files if the two jobs have the same sequence number. For example, consider the following two BATCH streams:

```
$FOR/OBJ A      $FOR/OBJ A
$FOR/OBJ B      $CALL C
$LINK/RUN       $FOR/OBJ B
                $LINK/RUN
```

The called BATCH file (C.BAT) contains the following:

```
$JOB
$FOR/OBJ A1
$FOR/OBJ B1
$LINK/RUN
$EOJ
```

The temporary object files C.BAT generates change the behavior of the previous two BATCH statement sequences. The first temporary file created by C.BAT (000000.OBJ) supersedes the temporary file produced by the first $FORTRAN command (000000.OBJ). You can avoid this situation by giving the BATCH job C.BAT a unique sequence number (see Section A.4.22).

A.4.3 $CHAIN

The $CHAIN command transfers control to a named BATCH control file but does not return to the input stream that executed the $CHAIN command. The syntax of the $CHAIN command is:

$CHAIN filespec[!comments]

BATCH does not permit options in the $CHAIN command. If you specify .CTL as the file type, BATCH assumes a precompiled BATCH control file. If you do not specify a file type, BATCH assumes .BAT and compiles the chained BATCH stream before execution.

A $EOJ command should always follow the $CHAIN command in the BATCH stream.

NOTE

The values of BATCH run-time variables remain constant across a $CALL, $CHAIN, or return from call. See Section A.5.2.2 for a description of these variables.

Use the $CHAIN command to transfer control to programs that you need to run only once at the end of a BATCH stream. For example, you could use the following BATCH program (PRINT.BAT) to print and then delete all temporary listing files generated during the current BATCH job.

```
$JOB                        !PRINT ALL LIST FILES
$PRINT/DELETE *.LST
$EOJ
```

You could then run PRINT.BAT with the $CHAIN command as follows:

```
$JOB
$MACRO/RUN                  A ALST/LIST
$MACRO/RUN                  B BLST/LIST
$CHAIN PRINT
$EOJ
```

A.4.4 $COPY

The $COPY command copies files in image mode from one device to another. You can use the wild card construction (see Section A.2.2.3) in the input and output file specifications. You can concatenate several input files to form one

output file (as long as the output specification does not contain a wild card). The $COPY command has the following syntax:

$COPY[/option] output-filespec[...,output-filespec]/OUTPUT-
input-filespec[...,input-filespec][/INPUT][!comments]

where:

/option	indicates options that you can append to the $COPY command
	/DELETE Deletes input files after the copy operation.
	/NODELETE Does not delete input files after the copy operation.
output-filespec	represents an output file; you must specify a file type
/OUTPUT	indicates that a file specification is for an output file
input-filespec	represents a file to be copied
	(BATCH copies files to the output file in the order that you list them, except when you use wildcards.)
/INPUT	indicates that a file specification is for an input file; if you do not specify an option, BATCH assumes INPUT

The following are examples of the $COPY command:

```
$COPY *.BAS/OUTPUT DT1:*.BAS
```

This command copies all files with the file type .BAS from the DECtape on unit 1 to the default storage device DK:.

```
$COPY FILE2.FOR/OUTPUT FILE0.FOR+FILE1.FOR
```

This command merges the input files FILE0.FOR and FILE1.FOR to form one file called FILE2.FOR and stores FILE2.FOR on device DK:.

```
$COPY *.*/OUT DT0:*.FOR, DT1:*.*/OUT DT0:*.*
```

This command copies all files with the file type .FOR from DT0: to DK: and all files on DT0: to DT1:.

A.4.5 $CREATE

The $CREATE command generates a file from data records that follow the $CREATE command in the input stream. An error occurs if the data does not immediately follow the $CREATE command. You cannot precede the data records with a $DATA command.

You can follow the $CREATE data with a $EOD command to signify the end of data, or you can use any other BATCH control statement to indicate end of data and initiate a new function. The $CREATE command has the following syntax:

$CREATE[/option...] filespec [!comments]

where:

/option indicates an option you can append to the $CREATE command. The options are:

- /DOLLARS — Indicates that the data following this command can have a $ in the first character position of a line.
- /NODOLLARS — Indicates that a $ cannot be in the first character position of a line.
- /LIST — Writes data image lines to the log file.
- /NOLIST — Does not write data image lines to the log file. If you specify $JOB/LIST, BATCH ignores this option.

filespec represents the file you want to create

NOTE

If you use the /DOLLARS option, you must follow the last data record with a $EOD command (see Table A–1).

The following is an example of the $CREATE command:

```
$CREATE/LIST PROG.FOR
FORTRAN source file
$EOD
```

The data records following the $CREATE command become a new file (PROG.FOR) on the default device (DK:). BATCH generates a listing on logical device LOG:.

A.4.6 $DATA

Use the $DATA command to include data records in the input stream. Data you include in this manner needs no file name. BATCH transfers the data to the appropriate program as though it were input from the console terminal. For example, you can follow the $RUN command for a particular program by a $DATA command and the data records for the program to process. The data records must be valid data for the program that is to use them.

The $DATA command has the following syntax:

$DATA[/option...] [!comments]

Four options that you can use with the $DATA command are as follows:

/DOLLARS	Indicates that the data following this command can have a $ in the first character position of a line.
/NODOLLARS	Indicates that a $ cannot be in the first character position of a line.
/LIST	Writes data image lines to the log file.
/NOLIST	Does not write data images to the log file. If you specify $JOB/LIST, BATCH ignores this option.

NOTE

Any command beginning with a $ normally follows the last data record. However, if you specify $DATA/DOLLARS, you must follow the last data record with $EOD.

The following example shows data entered into a BASIC program (TEST1.BAS).

```
$BASIC/RUN TEST1.BAS
$DATA
25,75,125,146
180,210,520,874
$EOD
```

A.4.6.1 Using $DATA with FORTRAN Programs — When you use the $DATA command to provide input to a FORTRAN program, you must insert a CTRL/Z into the BATCH file after the last data line and before $EOD (or before the next BATCH command if you do not use $EOD). This procedure permits FORTRAN to properly detect an end-of-file after it reads the last data line. For example:

```
$FORTRAN/RUN  A.FOR
$DATA
1
2
3
^Z <RET>  <LF>
$EOD
$RUN PIP
```

The above program reads three numbers from the input stream and then detects an end-of-file when it attempts to read a fourth number. If you include an END=n statement in your FORTRAN program, statement n gets control when the end-of-file is detected. If the CTRL/Z <RET> <LF> is not present, the program aborts when it reaches $EOD and never executes the END=n statement.

A.4.7 $DELETE

Use the $DELETE command to delete files from the device you specify. This command has the syntax:

$DELETE filespec[...,filespec][!comments]

where:

filespec represents the name of a file to be deleted

The following example deletes all files named TEST1 on the default device DK:.

```
$DELETE TEST1.*
```

The following example deletes all files with .FOR file types on DT1:, then deletes all files with .MAC file types on DK:.

```
$DELETE DT1:*.FOR;*.MAC
```

A.4.8 $DIRECTORY

The $DIRECTORY command outputs a directory of the device you specify to a listing file. If you do not specify a listing file, the listing goes to the BATCH log file. This command has the syntax:

$DIRECTORY [filespec/LIST] [filespec[...,filespec]][/INPUT][!comments]

where:

filespec/LIST indicates the name of the directory listing file

filespec/INPUT indicates the input files to be included in the directory (default)

The following command outputs a directory of the device DK: to the BATCH log file.

```
$DIRECTORY
```

This next command creates on the device DK: a directory file (FOR.DIR) that contains the names, lengths, and dates of creation of all FORTRAN source files on that device.

```
$DIRECTORY FOR.DIR/LIST *.FOR
```

A.4.9 $DISMOUNT

The $DISMOUNT command removes the logical device name assigned by a $MOUNT command. When BATCH encounters $DISMOUNT while executing a job, it prints the entire $DISMOUNT command line on the console terminal. This message tells the operator which device to unload. This command has the syntax:

$DISMOUNT[/option] logical-device-name:[/LOGICAL] [!comments]

where:

/option — indicates an option you can append to the $DISMOUNT command. The options are:

- /WAIT — Indicates that the job must pause until the operator enters a response. If you do not specify either /WAIT or /NOWAIT, BATCH assumes /WAIT. BATCH rings a bell at the terminal, prints the physical device name to be dismounted followed by a question mark (?), and waits for a response. (At this point you can enter input to the BATCH handler. See Section A.7.3.)
- /NOWAIT — Does not pause for operator response; BATCH prints the physical device name to be dismounted.

logical-device-name: — is the logical device name to be deassigned from the physical device

/LOGICAL — identifies the device specification as a logical device name

The following example instructs the operator to dismount the physical device with the logical device name OUT: and removes the logical assignment of device OUT:. In this example, OUT: is DT0:. The operator dismounts DT0: and then types a carriage return.

```
$DISMOUNT/WAIT OUT:/LOGICAL
DT0?
```

A.4.10 $EOD

The $EOD command indicates the end-of-data record or the end of a source program in the job stream. The syntax of this command is:

$EOD [!comments]

The $EOD command can signal the end of data associated with any of the following commands:

$BASIC	$FORTRAN
$CREATE	$MACRO
$DATA	

In the following example, the $EOD command indicates the end of a source program that is to be compiled, linked, and executed.

```
$FORTRAN/RUN
source program
$EOD
```

A.4.11 $EOJ

The $EOJ command indicates the end of a job. This command must be the last statement in every BATCH job. The command has the following syntax:

$EOJ [!comments]

If BATCH encounters a $JOB command, a $SEQUENCE command, or a physical end-of-file in the input stream before $EOJ, an error message appears in the log file.

NOTE

Make sure that the $EOJ command is the last line in a BATCH file.

A.4.12 $FORTRAN

The $FORTRAN command calls the FORTRAN compiler to compile a source program. Optionally, this command can provide printed listings or list files and can produce a link map in the listing. The $FORTRAN command has the following syntax:

$FORTRAN[/option...] [source-filespec[/option]] [filespec/OBJECT]-
[filespec/LIST] [filespec/EXECUTE]-
[filespec/MAP] [filespec/LIBRARY] [!comments]

where:

/option indicates an option you can append to the $FORTRAN command. The options are as follows:

/RUN	Indicates that FORTRAN is to compile the source program, link it with the default library, and execute it. The default library is SYSLIB.OBJ. You can change it with the $LIBRARY command.
/NORUN	Compiles the program only.
/OBJECT	Produces a temporary object file.
/NOOBJECT	Does not produce a temporary object file.
/LIST	Produces a list file on the listing device (LST:).
/NOLIST	Does not produce a list file.
/MAP	Produces a link map on the listing device (LST:).
/NOMAP	Does not create a MAP file.

/DOLLARS Indicates that the data following this command can have a $ in the first character position of a line.

/NODOLLARS Indicates that a $ cannot be in the first character position of a line.

source-filespec indicates the device, file name, and file type of the FORTRAN source file. If you do not specify the file name, the $FORTRAN source statements must immediately follow the $FORTRAN command in the input stream; BATCH generates a temporary source file that it deletes after FORTRAN compiles the temporary source file (see Section A.2.5).

You can terminate the source program included after a $FORTRAN statement by either a $EOD command or by any other BATCH command. If, however, you use dollar signs in the first position in the source program, you must enter the source program with $CREATE/DOLLARS. In this case, you cannot use $FORTRAN/DOLLARS

/option represents an option that can have one of the following values:

/FORTRAN Indicates that the file name you specify is a FORTRAN source program. BATCH assumes that any file name with no option appended is the name of a source file.

/SOURCE Performs the same function as /FORTRAN.

/INPUT Performs the same function as /FORTRAN.

filespec/OBJECT indicates the device, file name, and file type of the object file produced by compilation. The object file remains on the device you specify after the job finishes. You must follow the object file specification, if you include it, with the /OBJECT option.

If you omit the object file specification but specify $FORTRAN/OBJECT, BATCH creates a temporary object file. BATCH includes this temporary file in any $LINK operations that follow it in the job, and deletes it after the link operation

filespec/LIST — indicates the name you assign to the list file created by the compiler. BATCH does not automatically print the list file if you assign LST: to a file-structured device, but you can list it using the $PRINT command. Follow the list file specification with the /LIST option

filespec/EXECUTE — indicates the name you assign to a memory image file. Follow the memory image file specification with the /EXECUTE option. If you do not include this field, BATCH generates a temporary memory image file (see Section A.2.5) and then deletes the temporary file

filespec/MAP — indicates the name you assign to the link map file created by the linker. Follow the map specification with the /MAP option

filespec/LIBRARY — indicates that BATCH must include the file you specify in the link procedure as a library before SYSLIB.OBJ. The file must be a library file (produced by the RT–11 librarian). Follow the library specification with the /LIBRARY option

The following command calls FORTRAN to compile and execute a source program named PROGA.FOR.

```
FORTRAN/RUN PROGA.FOR
```

The next command sequence compiles the FORTRAN program but does not produce an object file. BATCH creates a temporary listing file on LST:.

```
$FORTRAN/NOOBJ/LIST
source program
$EOD
```

NOTE

See Section A.4.6.1 for instructions on using the $DATA command with FORTRAN programs.

A.4.13 $JOB

The $JOB command indicates the beginning of a job. Each job must have its own $JOB command. This command has the following syntax:

$JOB[/option...] [!comments]

BATCH allows the following options in the $JOB command:

/BANNER — Prints a header (a repetition of the $JOB line or card) on the log file.

/NOBANNER	Does not print a job header.
/LIST	Writes data image lines that are contained in the job stream to the log file.
/NOLIST	Writes data image lines to the log file only when a /LIST option exists on a $BASIC, $CREATE, or $DATA command that has data lines following it.
/RT11	If no $ appears in column 1 when BATCH expects one, BATCH assumes that the line or card is an RT–11 mode command (see Section A.5).
/NORT11	Does not process RT–11 mode commands.
/TIME	Writes the time of day to the log file when BATCH executes command lines (except $DATA command lines).
/NOTIME	Does not write the time of day.
/UNIQUE	Checks for unique spelling of options and keynames. When you use this option, you can abbreviate commands and options to the fewest number of characters that still make their names unique. For example, you can abbreviate the /DOLLARS option to /DO since no other option begins with the characters DO.
/NOUNIQUE	Checks only for normal option and keyname spellings.

End each job with a $EOJ command if you want to run it. If an input stream consists of more than one job, BATCH automatically terminates one job when it encounters the $JOB command for the next job. BATCH will never run a job terminated with another $JOB command; instead, an error message will appear in the log.

The following $JOB command writes the time of day to the log file before BATCH executes each command beginning with a $. It also accepts unique abbreviations of BATCH commands and options.

```
$JOB/TIME/UNIQUE
```

A.4.14 $LIBRARY

The $LIBRARY command lets you specify a list of library files for inclusion in FORTRAN links or other link operations that have the /LIBRARY option. By default, the list of libraries contains only SYSLIB.OBJ, the RT–11 system library. This command has the syntax:

$LIBRARY filespec [!comments]

or

$LIBRARY filespec + SYSLIB [!comments]

where:

filespec represents a library file; the default file type is .OBJ

SYSLIB is the RT–11 system library that you create at system generation

Libraries are linked in order of their appearance in the $LIBRARY command.

The following example shows two libraries (LIB1.OBJ and LIB2.OBJ) that are included in FORTRAN links before SYSLIB.OBJ.

```
$LIBRARY LIB1.OBJ+LIB2.OBJ+SYSLIB.OBJ
```

A.4.15 $LINK

Use the $LINK command to produce memory image files from object files. This command links any files you may specify with any temporary object files created since the last link or link-and-go operation.

Temporary object files are those files you create as a result of a $FORTRAN or $MACRO command without naming an object file (with the /OBJECT option) by suppressing an object file (with the /NOOBJECT option). Create permanent object files by using the /OBJECT option on a $FORTRAN or $MACRO file descriptor.

BATCH links files in the following order:

1. Temporary files — in the order in which they were compiled
2. Permanent files — in the order in which they are specified in the $LINK command
3. Any library specified by the $LINK command — provided that unresolved references remain
4. The default library list — if you specified $LINK/LIBRARY

The syntax for this command is:

$LINK[/option...] [filespec/OBJECT] [filespec/LIBRARY]-
[filespec/MAP] [filespec/EXECUTE] [!comments]

where:

/option indicates an option that you can append to the $LINK command. The options are as follows:

/LIBRARY Includes the RT–11 system library (SYSLIB.OBJ) and any default libraries specified in the $LIBRARY command in this $LINK operation. Use this

option when the files being linked do not include any temporary FORTRAN object files. You can also use it when you specify $FORTRAN without the /RUN or /MAP option, but want to search the default library list for unresolved references.

/NOLIBRARY Does not include the default libraries.

/MAP Produces a temporary load map on the listing device (LST:).

/NOMAP Does not produce a map file.

/OBJECT Includes temporary object files in the link. If you specify neither /OBJECT nor /NOOBJECT, BATCH assumes $LINK/OBJECT.

/NOOBJECT Does not include temporary files in the link.

/RUN Executes the memory image files associated with this $LINK command when the link is complete.

/NORUN Only links the program and does not execute it.

filespec/OBJECT indicates the name of the object file BATCH must link; if you do not specify /OBJECT, BATCH assumes it as the default

filespec/LIBRARY indicates that the file you specify is to be included in the link procedure as a library; the file you specify must be a library file (produced by the RT–11 librarian)

filespec/MAP indicates the load map file BATCH must create as a result of the $LINK command

filespec/EXECUTE indicates the memory image file BATCH must create as a result of the $LINK command

The following command links all temporary object files created since the last $LINK command, or the last $FORTRAN/OBJ or $MACRO/OBJ command.

```
$LINK/RUN
```

The next command links the temporary files and the object files PROG1.OBJ and PROG2.OBJ to form a memory image file named PROGA.SAV. It also creates and outputs a temporary map file.

```
$LINK/MAP PROG1.OBJ+PROG2.OBJ/OBJ PROGA.SAV/EXE
```

A.4.16 $MACRO

The $MACRO command calls the MACRO assembler to assemble a source program and, optionally, to provide printed listings or list files. You must specify any MACRO listing directives in the source program; you cannot enter them at BATCH command level.

The $MACRO command has the following syntax:

$MACRO[/option...] [source-filespec[/option]] [filespec/OBJECT]-
[filespec/LIST] [filespec/MAP] [filespec/LIBRARY]-
[filespec/EXECUTE] [!comments]

where:

/option	indicates an option you can append to the $MACRO command. The options are as follows:

/RUN	Assembles, links, and runs the source program.
/NORUN	Only assembles the source program.
/OBJECT	Produces a temporary object file.
/NOOBJECT	Does not produce a temporary object file.
/LIST	Produces a listing file on the listing device (LST:).
/NOLIST	Does not produce a list file.
/CREF	Produces a cross-reference listing during assembly.
/NOCREF	Does not produce a cross-reference listing during assembly.
/MAP	Produces a link map as part of the listing file on LST:.
/NOMAP	Does not create a MAP file.
/DOLLARS	Indicates that the data following this command can have a $ in the first character position of a line.

/NODOLLARS Indicates that a $ cannot be in the first character position of a line.

/LIBRARY Includes the default library in the link operation.

/NOLIBRARY Does not include the default library in the link operation.

source-filespec indicates the name of the source file. If you do not specify a file name, the $MACRO source statements must immediately follow the $MACRO command in the input stream.

You can terminate the source program you include after a $MACRO statement with either a $EOD command or any other BATCH command. If, however, you include dollar signs in the first position in the source program, use the $CREATE/DOLLARS command to enter the source program. In this case, you cannot use $MACRO/DOLLARS

/option can have one of the following values:

/MACRO Indicates that the file name you specify is a MACRO source program. BATCH assumes that any file name with no option appended is the name of a source file.

/SOURCE Performs the same function as /MACRO.

/INPUT Performs the same function as /MACRO.

filespec/OBJECT indicates the name you assign to the object file produced by compilation. The object file remains on the device you specify after the job finishes. If you include an object file specification, follow it with the /OBJECT option.

If you omit the object file specification but specify $MACRO/ OBJECT, BATCH creates a temporary object file. BATCH also includes the temporary object file in any $LINK operations that follow the $MACRO command in the job, and deletes it after the link operation (see Section A.2.5)

filespec/LIST — indicates the name you assign to the list file created by the assembler. BATCH does not print the list file if you assign LST: to a file-structured device, but you can list it using the $PRINT command. The /LIST option must follow the list file specification

filespec/MAP — indicates the file to which BATCH must output the storage map

filespec/LIBRARY — indicates that BATCH must include the file you specify in the link procedure as a library. The /LIBRARY option must follow the library file specification

filespec/EXECUTE — indicates the name you assign to a memory image file. The /EXECUTE option must follow the memory image file specification. If you do not include this field but do use $MACRO/RUN, BATCH generates and runs a temporary memory image file (see Section A.2.5)

The following $MACRO command assembles a program named PROG0.MAC, and creates a temporary object file and a temporary listing file.

```
$MACRO/LIST/OBJECT PROG0.MAC
```

A.4.17 $MESSAGE

Use the $MESSAGE command to issue a message to the operator at the console terminal. It provides a means for the job to communicate with the operator. The $MESSAGE command has the syntax:

$MESSAGE[/option] message [!comments]

where:

/option — indicates an option you can append to the $MESSAGE command. The options are:

- /WAIT — Indicates that the job is to pause until the operator either types a carriage return to continue or enters commands to the BATCH handler followed by a carriage return (see Section A.7.3).
- /NOWAIT — Does not pause for operator response.

message — is a string of characters that must fit on one console line. BATCH prints the message on the console

For example, if you include the following message in the input stream:

```
$MESSAGE/WAIT MOUNT SCRATCH TAPE ON MT0:
```

The message:

```
MOUNT SCRATCH TAPE ON MTO:
?
```

appears on the console terminal and a bell sounds. The operator mounts the tape and types carriage return to allow further processing of the job. (See Section A.7.3 for operator interaction with BATCH.)

NOTE

BATCH compresses multiple spaces and tabs in BATCH command lines; therefore, attempts to format $MESSAGE output with tabs or spaces may not provide you with the desired results.

A.4.18 $MOUNT

The $MOUNT command assigns a logical device name and other characteristics to a physical device. When BATCH encounters $MOUNT during the execution of a job, it prints the entire $MOUNT command line on the console terminal to notify the operator which volume to use.

The $MOUNT command has the syntax:

$MOUNT[/option...] physical-device-name:[/PHYSICAL][/VID=x]
[logical-device-name:/LOGICAL] [!comments]

where:

/option — indicates an option you can append to the $MOUNT command. The options are:

- /WAIT — Indicates that the job is to pause until the operator enters a response. If you do not specify either /WAIT or /NOWAIT, BATCH assumes /WAIT. BATCH rings a bell, prints the physical device name and a question mark (?), and waits for a response. (The response can consist of input for the BATCH handler; see Section A.7.3.)
- /NOWAIT — Does not pause for operator response. BATCH prints the name of the physical device to be mounted.
- /WRITE — Tells the operator to write-enable the volume.
- /NOWRITE — Tells the operator to write-protect the volume.

physical-device-name

is required and specifies the physical device name and an optional unit number followed by a colon (for example, DT1:). If you specify a device name without a unit number, the operator can enter one in response to the question mark printed by the $MOUNT command. If you want the operator to supply a unit number, do not use the /NOWAIT option because it assumes unit 0

/PHYSICAL — identifies the device specification as a physical unit specification. If you do not specify either /PHYSICAL or /LOGICAL, BATCH assumes /PHYSICAL

/VID=x
/VID="x" — provides volume identification. The volume identification is the name physically attached to the volume. Include it to help the operator locate the volume.

Use this option only on the physical device file specification. If *x* contains spaces, specify it as "x"

NOTE

This volume identification is only a visual check for the operator. Make the identification match the visual label on the volume, not the identification that you wrote onto the volume at initialization time with the INIT/VOLUMEID command.

logical-device-name/LOGICAL

is required to identify any logical device name you may assign to the device. The /LOGICAL option must follow the logical device name specification

The following command instructs the operator to select a DECtape unit and mount DECtape volume BAT01 on that unit, write-enabled. It informs the operator by printing:

```
$MOUNT/WAIT/WRITE DT:/VID=BAT01 2:/LOGICAL
DT?
```

The operator selects a unit, mounts DECtape volume BAT01, write-enabled, and responds to the question mark by typing the unit number (such as, 1) followed by a carriage return. BATCH assigns logical device name 2 to the physical device (in this case, DT1:) and proceeds.

If no unit number response is necessary, as this command shows,

```
$MOUNT/WAIT/WRITE DT1: 2:/LOGICAL
```

the operator responds with a carriage return after mounting the DECtape and write-enabling the device.

A.4.19 $PRINT

Use the $PRINT command to print the contents of the files you specify on the listing device (LST:). This command has the syntax:

$PRINT[/option] filespec [...,filespec][/INPUT] [!comments]

where:

/option indicates an option you can append to the $PRINT command. The options are:

/DELETE Deletes input files after printing.

/NODELETE Does not delete input files after printing.

filespec represents a file to be printed

/INPUT indicates that the file is an input file; BATCH assumes /INPUT if you omit it

The following command prints a listing of files with file type .MAC that are stored on default device DK:.

```
$PRINT *.MAC
```

The following example creates listing files for the programs A and B, prints the listing files, and then deletes them.

```
$MACRO A.MAC A/LIST
$MACRO B.MAC B/LIST
$PRINT/DELETE A.LST,B.LST
```

A.4.20 $RT11

The $RT11 command allows the BATCH job to communicate directly with the RT–11 system. DIGITAL recommends that you use RT–11 mode if you use BATCH. This command puts BATCH in RT–11 mode until BATCH encounters a line beginning with $. In RT–11 mode, BATCH interprets all data images as commands to the RT–11 monitor, to RT–11 system programs, or to the BATCH run-time system. The $RT11 command has the syntax:

$RT11 [!comments]

See Section A.5 for a complete description of the RT–11 mode.

A.4.21 $RUN

The $RUN command executes a program for which a memory image file (.SAV) was previously created. It can also run RT–11 system programs.

The $RUN command has the syntax:

$RUN filespec [!comments]

where:

filespec represents the file to be executed. If you omit the file type, BATCH assumes .SAV

For example, if DIR is on DK:, you can run DIR to print a directory listing:

```
$RUN DIR
$DATA
LP:=DK:/L
$EOD
```

A.4.22 $SEQUENCE

The $SEQUENCE command is an optional command. If you use it, it must immediately precede a $JOB command. The $SEQUENCE command assigns a job an arbitrary identification number. BATCH assigns the last three characters of a sequence number as the first three characters of a temporary listing or object file (see Section A.2.5). If a sequence number is less than three characters long, BATCH fills it with zeroes on the left.

The syntax of this command is:

$SEQUENCE id [!comments]

where:

id represents an unsigned decimal number that indicates the identification number of a job

The following are examples of the $SEQUENCE command:

```
$SEQUENCE 3           !SEQUENCE NUMBER IS 003
$JOB

$SEQUENCE 100         !SEQUENCE NUMBER IS 100
$JOB
```

A.4.23 Sample BATCH Stream

The following sample BATCH stream creates a MACRO program, assembles and links that program, and runs the memory image file. It then deletes the object, memory image, and source files it created and prints a directory of DK: showing the files the BATCH stream created.

```
$JOB
$MESSAGE          THIS IS AN EXAMPLE BATCH STREAM
$MESSAGE          NOW CREATE A MACRO PROGRAM
$CREATE/LIST      EXAMPL.MAC
.TITLE  EXAMPL FOR BATCH
        .MCALL  .PRINT,.EXIT
START:  .PRINT  #MESSAG
        .EXIT
```

(continued on next page)

```
MESSAG: .ASCIZ  /EXAMPLE MACRO PROGRAM FOR BATCH/
        .END    START
$EOD
$MACRO  EXAMPL EXAMPL/OBJECT EXAMPL/LIST    !ASSEMBLE
$LINK   EXAMPL EXAMPL/EXECUTE               !AND LINK
$PRINT/DELETE EXAMPL.LST
$MESSAGE        RUN THE MACRO PROGRAM
$RUN    EXAMPL                              !AND EXECUTE
$DELETE EXAMPL.OBJ+EXAMPL.SAV+EXAMPL.MAC
$MESSAGE        PRINT A DIRECTORY
$DIRECTORY      DK:EXAMPL.*
$MESSAGE        END OF THE EXAMPLE BATCH STREAM
$EOJ
```

To run this batch stream, type the following commands at the console. BATCH prints the messages.

```
.LOAD BA,LP
.ASSIGN LP: LOG
.ASSIGN LP: LST
.R BATCH
*EXAMPL
 THIS IS AN EXAMPLE BATCH STREAM
 NOW CREATE A MACRO PROGRAM
 RUN THE MACRO PROGRAM
 PRINT A DIRECTORY
 END OF THE EXAMPLE BATCH STREAM

END BATCH
.
```

The above sample BATCH stream produces the following log file on the line printer:

NOTE

The amount of free memory and the directory format are variable.

```
$JOB

$MESSAGE        THIS IS AN EXAMPLE BATCH STREAM

$MESSAGE        NOW CREATE A MACRO PROG.

$CREATE/LIST    EXAMPL.MAC

.TITLE  EXAMPLE FOR BATCH
        .MCALL  .PRINT,.EXIT
START:  .PRINT  #MESSAG
        .EXIT
MESSAG: .ASCIZ  /EXAMPLE MACRO PROGRAM FOR BATCH/
        .EVEN
        .END    START
0

$EOD

$MACRO  EXAMPL EXAMPL/OBJECT EXAMPL/LIST    !ASSEMBLE

ERRORS DETECTED:  0
```

```
EXAMPLE FOR BATCH       MACRO V03.00 21-JUN-77 00:05:29 PAGE 1

      1                                 .TITLE  EXAMPLE FOR BATCH
      2                                         .MCALL  .PRINT,.EXIT
      3 000000                          START:  .PRINT  #MESSAG
      4 000006                                  .EXIT
      5 000010     105     130     101  MESSAG: .ASCIZ  /EXAMPLE MACRO PROGRAM FOR BATCH/
        000013     115     120     114
        000016     105     040     115
        000021     101     103     122
        000024     117     040     120
        000027     122     117     107
        000032     122     101     115
        000035     040     106     117
        000040     122     040     102
        000043     101     124     103
        000046     110     000
      6                                 .EVEN
      7        000000'                  .END    START

EXAMPLE FOR BATCH       MACRO V03.00 21-JUN-77 00:05:29 PAGE 1-1
SYMBOL TABLE

MESSAG  000010R         START   000000R

. ABS.  000000     000
        000050     001
ERRORS DETECTED:  0

VIRTUAL MEMORY USED:  508 WORDS  ( 2 PAGES)
DYNAMIC MEMORY AVAILABLE FOR  48 PAGES
EXAMPL,EXAMPL=EXAMPL

$LINK   EXAMPL EXAMPL/EXECUTE              !AND LINK

$PRINT/DELETE EXAMPL.LST


$MESSAGE        RUN THE MACRO PROGRAM

$RUN    EXAMPL                                !AND EXECUTE

EXAMPLE MACRO PROGRAM FOR BATCH

$DELETE EXAMPL.OBJ+EXAMPL.SAV+EXAMPL.MAC


$MESSAGE        PRINT A DIRECTORY

$DIRECTORY      DK:EXAMPL.*

 21-JUN-77
EXAMPL.BAK      2 14-JUN-77       EXAMPL.BAT      2 21-JUN-77
EXAMPL.CTL      3 21-JUN-77
 3 FILES, 7 BLOCKS
 1903 FREE BLOCKS

$MESSAGE        END OF THE EXAMPLE BATCH STREAM

$EOJ
```

A.5 RT–11 Mode

RT–11 mode lets you enter commands to the RT–11 monitor or to system programs, and lets you create BATCH programs. You can enter RT–11 mode with either the $JOB/RT11 command or the $RT11 command. If you enter RT–11 mode with the $JOB/RT11 command, RT11 mode remains in effect until BATCH encounters the next $JOB command. If you enter RT–11 mode

with the $RT11 command, RT–11 mode is in effect until BATCH encounters a $ in the first position of the command line.

When the characters ., $, *, and tab or space appear in the first position of a line (or card column 1), they are control characters and indicate the following:

. command to the RT–11 monitor, for example,

```
.R PIP
```

* data line; any line not intended to go to the RT–11 monitor or to the BATCH run-time handler, such as a command to the RT–11 PIP program:

```
*FILE1.DAT/D
```

NOTE

BATCH does not pass the * as data to the program. Comment lines (!) cannot appear on data lines as BATCH would consider them as data.

$ BATCH command. It causes an exit from RT–11 mode if you entered RT–11 mode with the $RT11 command. For example:

```
$RT11                    !ENTER RT-11 MODE
.R PIP
*FILE1.DAT/D
$FORTRAN                 !LEAVE RT-11 MODE
```

space/tab separator to indicate a line directed to BATCH run-time handler. This separator is indicated by a <TAB> in the following descriptions.

A.5.1 Communicating with RT–11

The most common use of RT–11 mode is to send commands to the RT–11 monitor and to run system programs. For example, you can insert the following commands in the BATCH stream to run PIP and save backup copies of files on DECtape:

```
$RT11
.R PIP
*DT1:*.*=*.FOR
```

You must anticipate and include in the BATCH input stream responses that the called program requires, such as the Y response to DUP's *Are you sure?* query. Place a line in your BATCH file consisting of Y and RETURN or use the DUP /Y option to suppress the query. For example:

```
$RT11
.INITIALIZE RK1:
*Y
```

You can communicate directly with the RT–11 monitor by using the keyboard monitor commands that are described in Section 4.3. For example:

```
$RT11
.DELETE/NOQUERY DX1:*.MAC
```

This command deletes all files with a file type of .MAC from device DX1:.

You cannot mix BATCH standard commands with RT–11 mode data lines (lines beginning with an asterisk). For example, the proper way to do a $MOUNT within a sequence of RT–11 mode data commands is:

```
$JOB/RT11
.R MACRO
*A1=A1
*A2=A2
$MOUNT DT0:/PHYSICAL
.R MACRO
*B1=DT:B1
*B2=DT:B2
```

A.5.2 Creating RT–11 Mode BATCH Programs

Advanced system programmers can use RT–11 mode to create BATCH programs. These BATCH programs consist of standard RT–11 mode commands (monitor commands, data lines for input to system programs, etc.) plus special RT–11 mode commands. The BATCH run-time handler interprets these special commands to allow dynamic calculations and conditional execution of the RT–11 mode standard commands. The following can help you create BATCH programs and dynamically control their execution at run-time:

- Labels
- Variable modification:
  1. equating a variable to a constant or character (LET statement)
  2. passing the value of a variable to a program
  3. incrementing the value of a variable by 1
  4. conditional transfers on comparison of variable values with numeric or character values (IF and GOTO statements)
- Commands to control terminal I/O
- Other control characters
- Comments

A.5.2.1 Labels — You define labels in RT–11 mode to provide a symbolic means of referring to a specific location within a BATCH program. If present, a label must begin in the first character position, must be unique within the first six characters, and must terminate with a colon (:) and a carriage return/line feed combination.

A.5.2.2 Variables — A variable in RT–11 mode is a symbol representing a value that can change during program execution. The 26 variables BATCH permits in a BATCH program have the names A–Z; each variable requires one byte of physical storage. There are four ways to modify variables.

You can assign values to variables in a LET statement.

You can then test these values by an IF statement to control the direction of program execution.

Assign values to variables with a LET statement of the following form:

<TAB>LET x = "c

where:

x represents a variable name in the range A–Z

"c indicates the ASCII value of a character

For example:

```
<TAB>LET A="0
```

This example indicates that the value of variable A is the 7-bit ASCII value of the character 0 (60).

The LET statement can also specify an octal value in the form:

```
<TAB>LET A=n
```

where:

n represents an 8-bit signed octal value in the range 0–377. Positive numbers range from 0–177; negative numbers range from 200–377 (–200 to –1)

You can use variables to introduce control characters, such as ESCAPE, into a BATCH stream. For example, wherever 'A' appears in the following BATCH stream, BATCH substitutes the contents of variable A (the code for an ESCAPE):

```
$JOB/RT11
   LET A=33
   !A IS AN ESCAPE
.R EDIT
*EBFILE.MAC'A''A'
*R'A''A'
   !EDIT FILE TO CHANGE THE VERSION NUMBER TO 2
*GVERSION='A'DI2'A''A'
*EX'A''A'
```

Increment the value of a variable by 1 by placing a percentage sign (%) before the variable. For example:

```
<TAB>%A
```

This command indicates that BATCH must increase the unsigned contents of variable A by 1.

Indicate with an IF statement conditional transfers of control according to the value of a variable. The IF statement has the syntax:

<TAB>IF(x-"c) label1, label2, label3

or

<TAB>IF(x-n) label1, label2, label3

where:

x	represents the variable to be tested
"c	is the ASCII value to be compared with the contents of the variable
n	is an octal integer in the range 0–377
label1 label2 label3	represent the names of labels included in the BATCH stream

When BATCH evaluates the expression (x-"c) or (x-n), the BATCH run-time handler transfers control to:

- label1 if the value of the expression is less than zero
- label2 if the value of the expression is equal to zero
- label3 if the value of the expression is greater than zero

If you omit one of the labels, and the condition is met for the omitted label, control transfers to the line following the IF statement.

NOTE

Since this comparison is a signed byte comparison, 377 is considered to be –1.

The characters + and – allow you to control where BATCH begins searching for label1, label2, and label3. If you precede the label by a minus sign (–), BATCH starts the label search just after the $JOB command. If a plus sign (+) or no sign precedes the label, the label search starts after the IF statement. For example:

```
<TAB>IF(B-"9) -LOOP, LOOP1,
```

This statement transfers program control to the label LOOP following the $JOB command if the contents of variable B are less than the ASCII value of 9. It transfers control to the label LOOP1 following the IF statement if B is equal to ASCII 9. If the contents of variable B are greater than the ASCII value of 9, program control goes to the next BATCH statement in sequence.

The GOTO statement unconditionally transfers program control to a label you specify as the argument of the statement. You can use one of the following three forms of this statement:

<TAB>GOTO label	transfers control to the first occurrence of label that appears after this GOTO statement in the BATCH stream
<TAB>GOTO +label	same as GOTO label
<TAB>GOTO -label	transfers control to the first occurrence of label that appears after the $JOB command

The following GOTO statement transfers control unconditionally to the next label LOOP if such a label appears in the BATCH stream following the GOTO statement.

```
<TAB>GOTO LOOP
```

NOTE

If BATCH cannot find a label (for example, you unintentionally omit a minus sign) the BATCH handler searches until it reaches the end of the .CTL file and ends the job.

A.5.2.3 Terminal I/O Control — You can issue commands directly to the BATCH run-time handler to control logging console terminal input and output. If you do not enter any of the following commands, BATCH assumes TTYOUT (this includes indirect command files).

<TAB>NOTTY	Does not write terminal input and output to the log file. Comments to the log are still logged.
<TAB>TTYIN	Writes only terminal input to the log file.
<TAB>TTYIO	Writes terminal input and output to the log file. (You should enter this command if using RT–11 mode so that RT–11 mode commands go to the log file.)
<TAB>TTYOUT	Writes only terminal output to the log file (default).

A.5.2.4 Other Control Characters — The system permits other control characters in an RT–11 mode command that begins with a period (.) or an asterisk (*). Following are these control characters and their meanings:

'text' command to BATCH run-time handler, where text can be one of the following:

CTY	Accepts input from the console terminal; notifies the operator that action is required by ringing a bell and printing a question mark (?).
FF	Outputs the current log buffer.
NL	Inserts a new line (line feed) in the BATCH stream.

x	Inserts the contents of a variable where *x* is an alphanumeric variable in the range A through Z. It indicates that BATCH should insert the contents of the variable as an ASCII character at this place in the command string.
"message"	Directs the message to the console terminal.

The following commands allow the operator to enter the name of a MACRO program to be assembled. The BATCH stream contains:

```
$JOB/RT11
.R MACRO
*'"ENTER MACRO COMMAND STRING"'CTY'
```

The operator receives the following message at the terminal and types a response, followed by carriage return; BATCH processing continues.

```
ENTER MACRO COMMAND STRING
?FILE,FILE=FILE
```

To run the same BATCH file on several systems with different configurations you need to assign a device dynamically. The following RT–11 mode command lets you request that the listing device name be entered by the operator.

```
.ASSIGN '"PLEASE TYPE LST DEVICE NAME"'CTY'LST
```

The operator receives the message and responds with the device to be used as the listing device (DT2:).

```
PLEASE TYPE LST DEVICE NAME
?DT2:
```

A.5.2.5 Comments — You can include comments in RT–11 mode as separate comment statements. Include comments by typing a separator followed by a ! and the comment. For example:

```
<TAB>!OPERATOR ACTION IS REQUESTED IN THIS JOB. BE PREPARED.
```

A.5.3 RT–11 Mode Examples

The following are examples of BATCH programs using the RT–11 mode.

This BATCH program assembles, lists, and maps 10 programs with only 12 BATCH commands.

```
$JOB/RT11     !ASSEMBLE, LIST, MAP PROG0 to PROG9
   TTYIO
   !WRITE TERMINAL I/O TO THE LOG FILE
   LET N="0
   !START AT FILE PROG0
```

(continued on next page)

```
LOOP:
.R MACRO
*PROG'N',LOG:/C=PROG'N'/N:TTM
.R LINK
*,LOG:=PROG'N'
   %N
   !INCREMENT VARIABLE N
   IF(N-"9)-LOOP,-LOOP,END
   !TEST FOR END
END:
$EOJ
```

The following program lets you set up a master control stream to run several BATCH jobs with one call to BATCH. First set up a BATCH job (INIT.BAT) that performs a $CHAIN to the master control stream:

```
$JOB/RT11
   LET I="0
   !INITIALIZE INDEX
$CHAIN MASTER        !GO TO MASTER
$EOJ
```

The following is the master control stream (MASTER.BAT) to which INIT chains.

```
$JOB/RT11            !MASTER CONTROL STREAM
   %I
   !BUMP INDEX BY 1
   IF(I-"7),,END
.R BATCH
   !THIS IS A $CHAIN
*JOB'I'
   !RUNS JOB1-JOB7
END:
$MESSAGE END OF BATCH RUN
$EOJ
```

Each job MASTER.BAT will run must contain the following:

```
$JOB
   !BATCH COMMANDS
$CHAIN MASTER
$EOJ
```

Activate the master control stream by calling BATCH as follows:

```
.R BATCH
*INIT
```

A.6 Creating BATCH Programs on Punched Cards

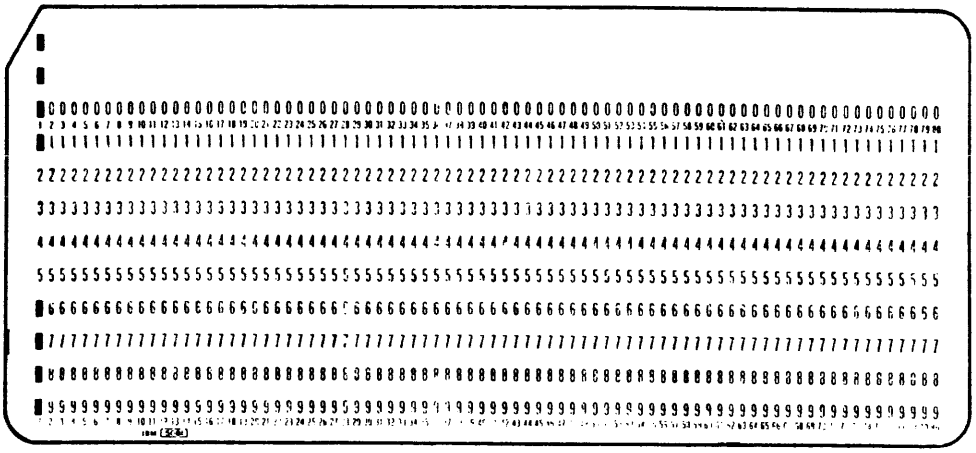
To create a BATCH program on punched cards, punch into the cards the commands described in Section A.4. Each command line occupies a single punched card. Only one card, the EOF card, is different from the standard BATCH commands. The EOF (end-of-file) card terminates the list of jobs from the card reader.

To create the EOF card, hold the MULT PCH key on the keypunch keyboard while typing the following characters:

- & 0 1 6 7 8 9

This procedure produces an EOF card with holes punched in the first column (see Figure A–1).

Figure A–1: EOF Card



To run multiple jobs from the card reader, simply combine the jobs into a single card deck. Make sure that each job has its own $JOB and $EOJ card, and then follow the last $EOJ card with two EOF cards.

Although in general, you terminate BATCH jobs on cards by placing two EOF cards after the last $EOJ card, some card readers may require that you type \F followed by a carriage return. Put two EOF cards and a blank card in the reader and make sure that the card reader is ready. Note that a small card deck (less than 512 characters) may require more than two EOF cards to terminate the deck.

A.7 Operating Procedures

A.7.1 Loading BATCH

After you bootstrap the RT–11 system and enter the date and time, you must make the BATCH run-time handler resident by typing the RT–11 LOAD command as follows:

```
.LOAD BA:
```

You detach and unload the BATCH run-time handler with the /U option in the BATCH compiler command line (see Section A.7.2).

NOTE

If BATCH crashes, you must unload BATCH with the UNLOAD command and then reload BATCH with the LOAD command. This ensures that the BATCH handler is properly initialized when you rerun BATCH.

You must make the BATCH log device and list device resident unless the log or list device is SY:, or unless it is a device for which the handler is already resident. Load the log device by typing:

.LOAD log-device

where:

log-device represents the device to which BATCH must write the log file

For example:

```
.LOAD LP:
```

You can, of course, load device handlers with a single LOAD command. For example:

```
.LOAD BA:,LP:
```

You must then assign the logical device name LOG to the log device. Use the RT–11 monitor ASSIGN command in the form:

.ASSIGN log-device LOG

For example, if LP: is the log device, type:

```
.ASSIGN LP LOG
```

Then assign the logical device name LST: using the RT–11 ASSIGN command in the form:

.ASSIGN list-device LST

where:

list-device represents the physical device BATCH must use for listings

If, for example, you want to produce listings on the line printer, type:

```
.ASSIGN LP LST
```

NOTE

Do not use the DEASSIGN command with no arguments in a BATCH program since it deassigns the log and list devices, possibly causing the BATCH job to terminate.

You must also make resident the BATCH run-time handler input device (compiler output device). If this device is already resident or is SY:, you do

not need to load it. For example, to load the DECtape handler as the input device, type:

```
•LOAD DT
```

If the input file to the BATCH compiler is on cards, load the card reader handler by typing:

```
•LOAD CR
```

NOTE

If input is on cards, you must use the RT–11 monitor SET command (before loading the handler) to specify CRLF and NOIMAGE modes. That is, the following command appends a carriage return/line feed combination to each card image.

```
•SET CR CRLF
```

The following command translates the card by packing card code into ASCII data, one column per byte.

```
•SET CR NOIMAGE
```

If card images do not properly translate to ASCII, you may have to change the card translation codes by using one of the following commands:

```
•SET CR CODE=29
```

or

```
•SET CR CODE=26
```

See Section 4.4.

A.7.2 Running BATCH

When you have loaded all necessary handlers, run the BATCH compiler as follows:

```
•R BATCH
```

BATCH responds by printing an asterisk (*) to indicate its readiness to accept commands. In response to the *, type the output file specifications for the control file followed by an equals sign. Then type the input file specifications for the BATCH file as follows:

[[output-filespec][,log-filespec][/option...] =]input-filespec[..., input-filespec][/option...]

where:

output-filespec is the BATCH compiler output device and file the BATCH run-time handler must use. The device you

specify must be random-access. Your BATCH job should not delete or move this file. Your BATCH job should avoid compressing the system volume with the SQUEEZE command or the DUP /S option. If you omit *output-filespec*, BATCH generates a file on the default device DK: with the same name as the first input file but with a .CTL file type. If you do not specify a file type in *output-filespec*, BATCH assumes .CTL

log-filespec — is the log file created by the BATCH run-time handler. If you do not specify a log device, BATCH assumes LOG: The device name you specify for *log-filespec* must be the same as you assign to LOG:.

You can change the size of a log file on a file-structured device from the default size of 64 (decimal) blocks. To make this change, enclose the required size in square brackets. For example:

```
*,FILE.LOG[10]=FILE
```

The default file type for the *log-filespec* is .LOG

input-filespec — represents an input file. If you do not specify a file type, BATCH assumes .BAT. If you specify a .CTL file, BATCH assumes a precompiled file that must be the only file in the input list

/option — is an option from the following list:

/N — Compiles but does not execute. This option creates a BATCH control file (.CTL), generates an ABORT JOB message at the beginning of the log file, and returns to the RT–11 monitor.

/T:n — If n = 0, sets the /NOTIME option as the default on the $JOB command. If n = 1, the default option on the $JOB command is /TIME.

/U — Indicates that the BATCH compiler must detach the BATCH run-time handler from the RT–11 monitor and unload the handler.

NOTE

You need not specify the RT–11 monitor UNLOAD BA command to remove the handler. Specifying /U to BATCH causes the handler to detach and unload.

/X — Indicates that the input is a precompiled BATCH program. Use this option when you do not specify the .CTL file type.

<RET> — Prints the version number of the BATCH compiler.

The following example calls BATCH to compile and execute three input files (PROG1.BAT, PROG2.BAT, PROG3.BAT) to generate on DK: the compiler output files, and to generate on LOG: a log file.

```
.R BATCH
*PROG1.BAT,PROG2.BAT,PROG3.BAT
```

The following commands print the version number of BATCH, then compile and run SYBILD.BAT.

```
.R BATCH
*<RET>
BATCH V04.00A
*SYBILD
```

The following commands compile PROTO.BAT to create PROTO.CTL but do not run the compiled BATCH stream.

```
.R BATCH
*PROTO/N
```

Type the following commands to unlink BA.SYS from the monitor and to unload it.

```
.R BATCH
*/U
```

The following commands compile FILE.BAT from magtape to create FILE.CTL on RK1:. They execute the compiled file and create a log file named FILE.LOG (of size 20) on LOG:.

```
.R BATCH
*RK1:FILE,FILE[20]=MT:FILE
```

The following commands execute a precompiled job called FILE.TST.

```
.R BATCH
*FILE.TST/X
```

The following commands execute a precompiled job called FILE.CTL.

```
.R BATCH
*FILE/X
```

The following commands accept input from the card reader to create a file called TEMP.CTL. BATCH stores this file on DK: and executes it.

```
.R BATCH
*CR:
```

The following commands accept input from the card reader to create a file called JOB.CTL. BATCH stores the file on DK: and executes it.

```
.R BATCH
*JOB=CR:
```

A.7.3 Communicating with BATCH Jobs

During the execution of a BATCH stream, BATCH can request the operator to service a peripheral device, to provide information, or to insert a command line into the BATCH stream. The operator does this by typing directives to the BATCH handler on the console terminal.

NOTE

These directives are equivalent to the compiler output that BATCH generates in the .CTL file. The .CTL file is an ASCII file that you can list by using the PRINT or TYPE commands or by running PIP.

These directives have the form:

\dir

where:

dir represents one of the directives listed in Table A–6

To use these directives, the operator must get control of the BATCH runtime handler. This can be achieved through a /WAIT or a CTY in the BATCH stream, or by typing a carriage return on the console terminal. If a carriage return is typed, the operator does not know exactly where the BATCH stream has been interrupted. When BATCH executes a command, it acknowledges the carriage return and prints a carriage return/line feed combination at the terminal. The operator can then enter a directive from Table A–6. The most useful directives are marked with an asterisk (*). Some directives are not particularly useful in this mode, but are listed to explain completely the BATCH compiler output.

Table A–6: Operator Directives to BATCH Run-Time Handler

Directive	Explanation
\@	Send the characters that follow to the console terminal.
*\A	Change the input source to be the console terminal.
*\B	Change the input source to be the BATCH stream.
*\C	Send the following characters to the log device.
*\D	Consider the following characters as user data.
*\E	Send the following characters to the RT–11 monitor.
*\F	Force the output of the current log block. If this directive is followed by any characters other than another BATCH backslash (\) directive, the BATCH job prints an error message and terminates. BATCH then returns control to the RT–11 monitor.
\G	Get characters from the console terminal until a carriage return is encountered.
\Hn	Help function to change the logging mode. *n* specifies the following: 0 Log only .TTYOUT and .PRINT 1 Log .TTYOUT, .PRINT, and .TTYIN 2 Do not log .TTYOUT, .PRINT, and .TTYIN 3 Log only .TTYIN
\Ivxlabel1? label2? label3?	IF statement that causes conditional transfer, where *v* is a variable name in the range A–Z; *x* is a value for the signed 8-bit comparison (v-x); and *label1, label2, label3* are 6-character labels to which control is transferred under certain conditions. (All labels must be six characters in length; if too short, pad with spaces.) If *v-x* is less than 0, control transfers to *label1*; if *v-x* is equal to 0, control goes to *label2*; if *v-x* is greater than 0, control goes to *label3*. The direction for the label search is indicated by *?*; if *?* is 0, the search begins at the beginning of this job; if *?* is 1, the label search begins after the IF statement.
\Jlabel?	Jump, unconditional transfer; where *label* is a 6-character label and *?* is 0 or 1. (All labels must be six characters in length; if too short, pad with spaces.) If *?* = 0, label is a backward reference; if *?* = 1, label is a forward reference.
\Kv0	Increment variable *v*, where *v* is a variable name in the range A–Z. Store the 8-bit number *n* in variable *v*.
\Kvln	Take the value in variable *v* and return it to the program (via .TTYIN).
\Kv2 \Llabel	Insert label as a 6-character alphanumeric string in the BATCH stream. (All labels must be six characters in length; if too short, pad with spaces.) Labels must not include backslash characters. Characters beyond six are ignored.

In the following example, the operator must interrupt the BATCH handler to enter information from the console. As a result of a /WAIT or 'CTY' in the BATCH stream, the following message appears at the terminal:

```
$MESSAGE/WAIT WRITE NECESSARY FILES TO DISK
```

To divert BATCH stream input from the current file to the console terminal, the operator types\E, enters commands to the RT–11 monitor, then types \B.

Control then returns to the BATCH stream. The following example illustrates this procedure.

```
.R BATCH
*NEXT
 WRITE NECESSARY FILES TO DISK
?\A\E

\ECOPY DT1:FILE.MAC RK:

 FILES COPIED:
DT1:FILE.MAC   TO RK:FILE.MAC

\E\F\B


END BATCH
```

The following BATCH program lets you make frequent edits to a file and list only the edits. First, create a BATCH program that assembles with a listing and link the file. This BATCH program, called COMPIL.BAT, contains:

```
$JOB/RT11
   TTYIO
   !WRITE TERMINAL I/O TO LOG FILE
.R MACRO
   !CALL THE MACRO ASSEMBLER
*FILE,FILE/C=FILE
$MESSAGE/WAIT OK TO TYPE EDIT COMMANDS
.R LINK
   !CALL THE RT-11 LINKER
*FILE,LOG:=FILE
$EOJ
```

At run time, you can insert commands into the BATCH stream from the console terminal. These commands search for the section of the listing file that has been edited, then list this section to the log. You must insert the command after the R MACRO command but before the R LINK command. The following example illustrates this procedure.

```
.R BATCH
*COMPIL
 OK TO TYPE EDIT COMMANDS
?\A\E

\ER EDIT

*ERFILE.LST$$
*EWFILE.SEC$$
*PRETRY:$=J$$
*\L$$
RETRY:  0                       ;HIGH ORDER BIT USED FOR *RESET IN PROGRESS FLAG
     49 000020  016705  177764                  MOV     RKCQE,R5        ;GET Q P
     50 000024  011502                          MOV     @R5,R2          ;R2 = BL
     51 000026  016504  000002                  MOV     2(R5),R4        ;R4 = UN
     52 000032  006204                          ASR     R4              ;ISOLATE
     53 000034  006204                          ASR     R4
     54 000036  006204                          ASR     R4
     55 000040  000304                          SWAB    R4
     56 000042  042704  017777                  BIC     #^C<160000>,R4
     57 000046  000404                          BR      2$              ;ENTER C
*EX$$

\E\C\B

END BATCH
```

A.7.4 Terminating BATCH

When BATCH terminates normally, it prints the following message and returns control to the RT–11 monitor:

```
END BATCH
```

To abort BATCH while it is executing a BATCH stream, interrupt the BATCH handler by typing a carriage return. When BATCH executes the next command after the carriage return, it prints a carriage return/line feed combination at the console terminal. You then gain control of the system. Type \F followed by a carriage return. The BATCH handler responds with the *FE* (forced exit) error message and writes the remainder of the log buffer. Control returns to the RT–11 monitor.

Typing two CTRL/Cs terminates BATCH immediately. Use two CTRL/Cs when BATCH is in a loop or when a long assembly is running. In these cases, BATCH responds slowly to your carriage return interrupt.

A.8 Differences Between RT–11 BATCH and RSX–11D BATCH

Some programmers run their RT–11 BATCH programs under RSX–11D. Note the differences between the two BATCH implementations listed in Table A–7. BATCH programs that run under both systems must be compatible with both RT–11 and RSX–11D BATCH.

Table A–7: Differences Between RT–11 and RSX–11D BATCH

Characteristic	RT–11	RSX–11D
File descriptors	filespec/option	SY:filnam.typ/option
Default listing file type	.LST(or .LIS)	.LIS
Executable file type	.SAV	.EXE
Incompatible commands	$BASIC $CALL $CHAIN $LIBRARY $RT11 $SEQUENCE	$MCR
Incompatible options	$COPY/DELETE $CREATE/DOLLARS $CREATE/LIST $DATA/DOLLARS $DATA/LIST	
	$DIR file/LIST	$DIR file/DIRECTORY
	$DISMOUNT/WAIT $DISMOUNT lun:/LOGICAL $FORTRAN/DOLLARS $FORTRAN/MAP	
	$JOB/BANNER	$JOB/NAME
	$JOB/LIST	$JOB/LIMIT
	$JOB/RT11	$JOB/MCR

(continued on next page)

Table A–7: Differences Between RT–11 and RSX–11D Batch (Cont.)

Characteristic	RT–11	RSX–11
Incompatible options	$JOB/TIME $JOB/UNIQUE $LINK/LIBRARY $LINK/OBJECT $MACRO/CREF $MACRO/DOLLARS $MACRO/LIBRARY $MACRO/MAP $MESSAGE/WAIT $MESSAGE/WRITE $PRINT/DELETE	$LINK/MCR
$DATA input	appears as if from input	appears as if from a file named FOR001.DAT
Logical device names	in $MOUNT and $DISMOUNT	Logical unit numbers only
$RUN	you must specify file name	RSX11DBAT.EXE is default

Appendix B
Monitor Command Abbreviations and System Utility Program Equivalents

This appendix provides two tables of correspondence (Tables B–1 and B–2) between the keyboard monitor commands with their options and the system utility programs with their options. Remember that the syntax you use to issue a keyboard monitor command is different from the syntax that the Command String Interpreter requires for input and output specifications for the system utility programs. Bear in mind that there are many differences between issuing a monitor command and running a utility program.

Table B–1 lists all the keyboard monitor commands and options. A dash under the corresponding system program or option column indicates that the command has no real system program equivalent, that the function is inherent in the keyboard monitor, or that the function is the default mode of operation. The minimum abbreviation for each command and option is in red.

Table B–1: Monitor Command/System Utility Program Equivalents

Monitor Command	Option	System Utility Program	Option
APL		R APL	—
ASSIGN		—	—
B		—	—
BASIC		R BASIC	—
BOOT		DUP	/O
	/WAIT	DUP	/W
	/FOREIGN	DUP	/Q
CLOSE		—	—
COMPILE		—	—
	/ALLOCATE:size	—	[n]
	/ALPHABETIZE	DIBOL	/A
	/CODE:type	FORTRAN	/I
	/CROSSREFERENCE [:type[...:type]]	MACRO,DIBOL	/C
	/DIAGNOSE	FORTRAN	/B
	/DIBOL	DIBOL	—
	/DISABLE:value[...:value]	MACRO	/D
	/ENABLE:value[...:value]	MACRO	/E
	/EXTEND	FORTRAN	/E
	/FORTRAN	FORTRAN	—

(continued on next page)

Table B–1: Monitor Command/System Utility Program Equivalents (Cont.)

Monitor Command	Option	System Utility Program	Option
COMPILE			
	/HEADER	FORTRAN	/O
	/I4	FORTRAN	/T
	/LIBRARY	MACRO	/M
	/LINENUMBERS	DIBOL,FORTRAN	—
	/NOLINENUMBERS	DIBOL, FORTRAN	/O /S
	/LIST[:filespec]	—	2nd output spec.
	/MACRO	MACRO	—
	/OBJECT[:filespec]	—	1st output spec.
	/NOOBJECT	—	null 1st output spec.
	/ONDEBUG	DIBOL,FORTRAN	/D
	/OPTIMIZE[:type]	FORTRAN	/P
	/NOOPTIMIZE	FORTRAN	/M
	/PASS:1	MACRO	/P
	/RECORD:length	FORTRAN	/R
	/SHOW:value	FORTRAN,MACRO	/L
	/NOSHOW:value	MACRO	/N
	/STATISTICS	FORTRAN	/A
	/SWAP	FORTRAN	—
	/NOSWAP	FORTRAN	/U
	/UNITS:n	FORTRAN	/N
	/VECTORS	FORTRAN	—
	/NOVECTORS	FORTRAN	/V
	/WARNINGS	FORTRAN	/W
	/NOWARNINGS	DIBOL, FORTRAN	/W —
COPY		PIP	—
	/ALLOCATE:size	—	[n]
	/ASCII	PIP,FILEX	/A
	/BINARY	PIP	/B
	/BOOT	DUP	/U
	/CONCATENATE	PIP	/U
	/DEVICE	DUP	/I
	/DOS	FILEX	/S
	/END:n	DUP	/E
	/EXCLUDE	PIP	/P
	/FILES	DUP	/F
	/IGNORE	PIP	/G
	/IMAGE	PIP,FILEX	/I
	/INTERCHANGE	FILEX	/U
	/LOG	PIP	/W
	/NOLOG	PIP	—
	/NEWFILES	PIP	/C
	/OWNER:[nnn,nnn]	FILEX	UIC

(continued on next page)

Table B–1: Monitor Command/System Utility Program Equivalents (Cont.)

Monitor Command	Option	System Utility Program	Option
COPY	/PACKED	FILEX	/P
	/POSITION:n	PIP	/M
	/PREDELETE	PIP	/O
	/QUERY	PIP,FILEX	/Q
	/NOQUERY	DUP	/Y
	/REPLACE	—	—
	/NOREPLACE	PIP	/N
	/SETDATE	PIP	/T
	/SLOWLY	PIP	/S
	/START:n	DUP	/G
	/SYSTEM	PIP	/Y
	/TOPS	FILEX	/T
	/WAIT	DUP	/W
		PIP	/E
CREATE		DUP	/C
	/ALLOCATE	DUP	[n]
	/EXTENSION	DUP	/T
	/START	DUP	/G
D		—	—
DATE		—	—
DEASSIGN		—	—
DELETE		—	—
	/DOS	FILEX	/S
	/ENTRY	QUEMAN	/M
	/EXCLUDE	PIP	/P
	/INTERCHANGE	FILEX	/U
	/LOG	PIP	/W
	/ENTRY	QUEMAN	1st output spec.
	/NEWFILES	PIP	/C
	/POSITION:n	PIP	/M
	/QUERY	PIP	/Q
	/NOQUERY	—	—
	/SYSTEM	PIP	/Y
	/WAIT	PIP	/E
DIBOL		R DIBOL	—
	/ALLOCATE:size	-	[n]
	/ALPHABETIZE	DIBOL	/A
	/CROSSREFERENCE	DIBOL	/O
	/LINENUMBERS	DIBOL	—
	/NOLINENUMBERS	DIBOL	/O
	/LIST[:filespec]	DIBOL	2nd output spec.
	/OBJECT[:filespec]	DIBOL	1st output spec.
	/NOOBJECT	DIBOL	null

(continued on next page)

Table B–1: Monitor Command/System Utility Program Equivalents (Cont.)

Monitor Command	Option	System Utility Program	Option
DIBOL			1st output spec.
	/ONDEBUG	DIBOL	/D
	/WARNINGS	DIBOL	—
	/NOWARNINGS	DIBOL	/W
DIFFERENCES		R SRCCOM	—
	/ALLOCATE:size		[n]
	/ALWAYS	BINCOM	/O
	/AUDITTRAIL	SRCCOM	/A
	/BINARY	BINCOM	—
	/BLANKLINES	SRCCOM	/B
	/BYTES	BINCOM	/B
	/CHANGEBAR	SRCCOM	/D
	/COMMENTS	SRCCOM	—
	/NOCOMMENTS	SRCCOM	/C
	/END[:n]	BINCOM	/E
	/FORMFEED	SRCCOM	/F
	/MATCH:n	SRCCOM	/L
	/OUTPUT:filespec	SRCCOM,	1st output spec.
		BINCOM	1st output spec.
	/PRINTER	SRCCOM,	LP: as 1st output spec.
		BINCOM	LP: as 1st output spec.
	/QUIET	BINCOM	/Q
	/SIPP:filespec	BINCOM	2nd output spec.
	/SLP	SRCCOM	/P
	/SPACES	SRCCOM	—
	/NOSPACES	SRCCOM	/S
	/START[:n]	BINCOM	/S
	/TERMINAL	SRCCOM,	TT: as 1st output spec.
		BINCOM	TT: as 1st output spec.
	/TRIM	SRCCOM	—
	/NOTRIM	SRCCOM	/T

(continued on next page)

Table B–1: Monitor Command/System Utility Program Equivalents (Cont.)

Monitor Command	Option	System Utility Program	Option
DIRECTORY		DIR	—
	/ALLOCATE:size	—	[n]
	/ALPHABETIZE	DIR	/A
	/BADBLOCKS	DUP	/K
	/BEFORE[date]	DIR	/K
	/BEGIN	DIR	/K
	/BLOCKS	DIR	/B
	/BRIEF	DIR, FILEX	/F
	/COLUMNS:n	DIR	/C
	/DATE[date]	DIR	/D
	/DELETED	DIR	/Q
	/DOS	FILEX	/S
	/END	DUP	/E
	/EXCLUDE	DIR	/P
	/FAST	DIR, FILEX	/F
	/FILES	DUP	/F
	/FREE	DIR	/M
	/FULL	DIR	/E
	/INTERCHANGE	FILEX	/U
	/NEWFILES	DIR	/D
	/OCTAL	DIR	/O
	/ORDER[:category]	DIR	/S
	/OUTPUT:filespec	DIR,	1st output file spec.
		FILEX	1st output file spec.
	/OWNER:[nnn,nnn]	FILEX	UIC
	/POSITION	DIR	/B
	/PRINTER	DIR,	LP: as 1st output spec.
		FILEX	LP: as 1st output file spec.
	/REVERSE	DIR	/R
	/SINCE[date]	DIR	/J
	/SORT[:category]	DIR	/S
	/START	DUP	/G
	/SUMMARY	DIR	/N
	/TERMINAL	DIR,	TT: as 1st output spec.

(continued on next page)

Table B–1: Monitor Command/System Utility Program Equivalents (Cont.)

Monitor Command	Option	System Utility Program	Option
DIRECTORY		FILEX	TT: as 1st output spec.
	/TOPS	FILEX	/T
	/VERIFY	DUP	/H
	/VOLUMEID[:ONLY]	DIR	/V
	/WAIT	DUP	/W
DUMP		R DUMP	—
	/ALLOCATE:size	—	—
	/ASCII	DUMP	—
	/NOASCII	DUMP	/N
	/BYTES	DUMP	/B
	/END:block	DUMP	/E
	/FOREIGN	DUMP	/T
	/IGNORE	DUMP	/G
	/ONLY:block	DUMP	/O
	/OUTPUT:filespec	DUMP	1st output file spec.
	/PRINTER	DUMP	LP: as 1st output spec.
	/RAD50	DUMP	/X
	/START:block	DUMP	/S
	/TERMINAL	DUMP	TT: as 1st output spec.
	/WORDS	DUMP	/W
E		—	—
EDIT		EDIT,TECO,KED, K52	EB
	/ALLOCATE:size	—	[n]
	/CREATE	EDITOR	—
	/EDIT	EDIT	—
	/EXECUTE	TECO	—
	/INSPECT	EDITOR	—
	/KED	KED	—
	/K52	K52	—
	/OUTPUT:filespec	EDITOR	—
	/TECO	TECO	—
EXECUTE		—	—
	/ALLOCATE:size	—	[n]
	/ALPHABETIZE	DIBOL	/A
	/BOTTOM:n	LINK	/B
	/CODE:n	FORTRAN	/I

(continued on next page)

Table B–1: Monitor Command/System Utility Program Equivalents (Cont.)

Monitor Command	Option	System Utility Program	Option
EXECUTE	/CROSSREFERENCE [:type[...:type]]	DIBOL,MACRO	/C
	/DEBUG[:filespec]	LINK	—
	/DIAGNOSE	FORTRAN	/B
	/DIBOL	DIBOL	—
	/DISABLE:value[...:value]	MACRO	/D
	/ENABLE:value[...:value]	MACRO	/E
	/EXECUTE[:filespec]	LINK	1st output file spec.
	/EXTEND	FORTRAN	/E
	/FORTRAN	FORTRAN	—
	/HEADER	FORTRAN	/O
	/I4	FORTRAN	/T
	/LIBRARY	MACRO	/M
	/LINENUMBERS	DIBOL, FORTRAN	—
	/NOLINENUMBERS	DIBOL, FORTRAN	/O /S
	/LINKLIBRARY:filespec	LINK	—
	/LIST[:filespec]	—	2nd output file spec.
	/MACRO	MACRO	—
	/MAP[:filespec]	LINK	2nd output file spec.
	/OBJECT[:filespec]	—	1st output file spec.
	/ONDEBUG	DIBOL, FORTRAN	/D
	/OPTIMIZE:type	FORTRAN	/P
	/NOOPTIMIZE	FORTRAN	/M
	/PASS:1	MACRO	/P
	/RECORD:length	FORTRAN	/R
	/RUN	RUN	—
	/NORUN	—	—
	/SHOW[:value]	FORTRAN, MACRO	/L
	/NOSHOW:value	MACRO	/N
	/STATISTICS	FORTRAN	/A
	/SWAP	FORTRAN	—
	/NOSWAP	FORTRAN	/U
	/UNITS:n	FORTRAN	/N
	/VECTORS	FORTRAN	—
	/NOVECTORS	FORTRAN	/V
	/WARNINGS	FORTRAN	/W
	/NOWARNINGS	DIBOL	/W
	/WIDE	LINK	/W

(continued on next page)

Table B–1: Monitor Command/System Utility Program Equivalents (Cont.)

Monitor Command	Option	System Utility Program	Option
FORMAT		—	—
	/PATTERN[:value]	FORMAT	/P
	/QUERY	FORMAT	—
	/NOQUERY	FORMAT	/Y
	/SINGLEDENSITY	FORMAT	/S
	/VERIFY[:only]	FORMAT	/V
	/WAIT	FORMAT	/W
FORTRAN		R FORTRAN	—
	/ALLOCATE:size	—	[n]
	/CODE:type	FORTRAN	/I
	/DIAGNOSE	FORTRAN	/B
	/EXTEND	FORTRAN	/E
	/HEADER	FORTRAN	/O
	/I4	FORTRAN	/T
	/LINENUMBERS	FORTRAN	—
	/NOLINENUMBERS	FORTRAN	/S
	/LIST[:filespec]	FORTRAN	2nd output file spec.
	/OBJECT[:filespec]	FORTRAN	1st output file spec.
	/NOOBJECT	FORTRAN	null 1st output spec.
	/ONDEBUG	FORTRAN	/D
	/OPTIMIZE:type	FORTRAN	/P
	/NOOPTIMIZE:type	FORTRAN	/M
	/RECORD:length	FORTRAN	/R
	/SHOW[:value]	FORTRAN	/L
	/STATISTICS	FORTRAN	/A
	/SWAP	FORTRAN	—
	/NOSWAP	FORTRAN	/U
	/UNITS:n	FORTRAN	/N
	/VECTORS	FORTRAN	—
	/NOVECTORS	FORTRAN	/V
	/WARNINGS	FORTRAN	/W
FRUN		—	—
	/BUFFER:n	—	—
	/PAUSE	—	—
	/TERMINAL:n	—	—
	/NAME:name	—	—
GET		—	—
GT OFF		—	—

(continued on next page)

Table B–1: Monitor Command/System Utility Program Equivalents (Cont.)

Monitor Command	Option	System Utility Program	Option
GT ON		—	—
	/L:n	—	—
	/T:n	—	—
HELP		—	—
	/PRINTER	—	—
	/TERMINAL	—	—
INITIALIZE		DUP	/Z
	/BADBLOCKS[:RET]	DUP	/B
	/DOS	FILEX	/S
	/FILE:filespec	DUP	—
	/INTERCHANGE	FILEX	/U
	/QUERY	DUP, FILEX	—
	/NOQUERY	DUP, FILEX	/Y
	/REPLACE[:RETAIN]	DUP	/R
	/RESTORE	DUP	/D
	/SEGMENTS:n	DUP	/N
	/VOLUMEID[:ONLY]	DUP	/V
	/WAIT	DUP	/W
INSTALL		—	—
LIBRARY		R LIBR	—
	/ALLOCATE:size	—	[n]
	/CREATE	LIBR	—
	/DELETE	LIBR	/D
	/EXTRACT	LIBR	/E
	/INSERT	LIBR	—
	/LIST[:filespec]	LIBR	2nd output file spec.
	/MACRO	LIBR	/M
	/OBJECT[:filespec]	LIBR	1st output file spec.
	/NOOBJECT	LIBR	null 1st output spec.
	/PROMPT	LIBR	//
	/REMOVE	LIBR	/G
	/REPLACE	LIBR	/R
	/UPDATE	LIBR	/U
LINK		R LINK	—
	/ALLOCATE:size	—	[n]
	/ALPHABETIZE	LINK	/A
	/BITMAP	LINK	—
	/NOBITMAP	—	/X
	/BOTTOM:n	LINK	/B
	/BOUNDARY:value	LINK	/Y

(continued on next page)

Table B–1: Monitor Command/System Utility Program Equivalents (Cont.)

Monitor Command	Option	System Utility Program	Option
LINK			
	/DEBUG[:filespec]	LINK	—
	/EXECUTE[:filespec]	LINK	1st output file spec.
	/NOEXECUTE	LINK	null 1st output spec.
	/EXTEND:n	LINK	/E
	/FILL:n	LINK	/Z
	/FOREGROUND[:stacksize]	LINK	/R
	/INCLUDE	LINK	/I
	/LDA	LINK	/L
	/LIBRARY:filespec	LINK	—
	/LINKLIBRARY:filespec	LINK	—
	/MAP[:filespec]	LINK	2nd output file spec.
	/PROMPT	LINK	//
	/ROUND:n	LINK	/U
	/RUN	LINK, RUN	—
	/SLOWLY	LINK	/S
	/STACK[:n]	LINK	/M
	/SYMBOLTABLE	LINK	3rd output file spec.
	/TOP	LINK	/H
	/TRANSFER[:n]	LINK	/T
	/WIDE	LINK	/W
	/XM	LINK	/V
LOAD		—	—
MACRO		R MACRO	—
	/ALLOCATE:size	—	[n]
	/CROSSREFERENCE [:type[...:type]]	MACRO	/C
	/DISABLE:value[...:value]	MACRO	/D
	/ENABLE:value[...:value]	MACRO	/E
	/LIBRARY	MACRO	/M
	/LIST[:filespec]	MACRO	2nd output file spec.
	/OBJECT[:filespec]	MACRO	1st output file spec.

(continued on next page)

Table B–1: Monitor Command/System Utility Program Equivalents (Cont.)

Monitor Command	Option	System Utility Program	Option
MACRO			
	/NOOBJECT	MACRO	null 1st output spec.
	/PASS:1	MACRO	/P
	/SHOW:value	MACRO	/L
	/NOSHOW:value	MACRO	/N
PRINT		—	—
	/COPIES:n	PIP, QUEMAN	/K
	/DELETE	PIP, QUEMAN	/D
	/FLAGPAGE	QUEMAN	/H
	/NOFLAGPAGE	QUEMAN	/N
	/LOG	PIP	/W
	/NOLOG	PIP	—
	/NAME	QUEMAN	1st output file spec.
	/NEWFILES	PIP	/C
	/PRINTER	PIP	LP: as 1st output spec.
	/PROMPT	QUEMAN	//
	/QUERY	PIP	/Q
	/WAIT	PIP	/E
R		—	—
REENTER		—	—
REMOVE		—	—
RENAME		PIP	—
	/LOG	PIP	/W
	/NOLOG	PIP	—
	/NEWFILES	PIP	/C
	/PROTECTION	PIP	/F
	/NOPROTECTION	PIP	/Z
	/QUERY	PIP	/Q
	/REPLACE	PIP	—
	/NOREPLACE	PIP	/N
	/SETDATE	PIP	/T
	/SYSTEM	PIP	/Y
	/WAIT	PIP	/E
RESET		—	—
RESUME		—	—
RUN		—	—
SAVE		—	—
SET		—	—

(continued on next page)

Table B–1: Monitor Command/System Utility Program Equivalents (Cont.)

Monitor Command	Option	System Utility Program	Option
SHOW	ALL	RESORC	/A
	CONFIGURATION	RESORC	/Z
	DEVICES	RESORC	/D
	ERRORS	ERROUT	—
	/ALL	ERROUT	/A
	/FILE:filespec	ERROUT	input file spec.
	/FROM[:date]	ERROUT	/F
	/OUTPUT:filespec	ERROUT	1st output file spec.
	/PRINTER	ERROUT	LP: as 1st output spec.
	/SUMMARY	ERROUT	/S
	/TERMINAL	ERROUT	TT: as 1st output spec.
	/TO[:date]	ERROUT	/T
	QUEUE	QUEMAN	/L
	TERMINALS	RESORC	/T
	JOBS	RESORC	/J
SQUEEZE		DUP	/S
	/OUTPUT:filespec	DUP	1st output file spec.
	/QUERY	DUP	—
	/NOQUERY	DUP	/Y
	/WAIT	DUP	/W
SRUN			
	/BUFFER:n	—	—
	/LEVEL:n	—	—
	/NAME:logical-jobname	—	—
	/PAUSE	—	—
	/TERMINAL:n	—	—
START		—	—
SUSPEND		—	—
TIME		—	—

(continued on next page)

Table B–1: Monitor Command/System Utility Program Equivalents (Cont.)

Monitor Command	Option	System Utility Program	Option
TYPE		—	—
	/COPIES:n	PIP	/K
	/DELETE	PIP	/D
	/LOG	PIP	/W
	/NOLOG	PIP	—
	/NEWFILES	PIP	/C
	/QUERY	PIP	/Q
	/WAIT	PIP	/E
UNLOAD		—	—

Table B–2 lists all the system programs and options with their keyboard monitor equivalents. Note that some system program options are inaccessible through any keyboard monitor command; each is marked by an asterisk.

In this table:

- the first column lists the system program
- the second column lists the system program's option
- the third column lists the keyboard monitor command by which the system program is invoked
- the fourth column lists the monitor equivalent of the system program option

Note also that some system programs are accessible through more than one keyboard monitor command (for example, DUP is accessed through BOOT, COPY, CREATE, DIRECTORY, INITIALIZE, and SQUEEZE).

Table B–2 System Program/Monitor Command Equivalents

System Program	Option	Keyboard Monitor	
		Command	Option
BINCOM			
	/B	DIFFERENCES/BINARY	/BYTES
	/E:n	DIFFERENCES/BINARY	/END[:end]
	/H	*	
	/O	DIFFERENCES/BINARY	/ALWAYS
	/Q	DIFFERENCES/BINARY	/QUIET
	/S:n	DIFFERENCES/BINARY	/START[:n]
DIR			
	/A	DIRECTORY	/ALPHABETIZE
	/B	DIRECTORY	/BLOCKS
	/C:n	DIRECTORY	/COLUMNS:n
	/D[:date]	DIRECTORY	/DATE[:date]
	/E	DIRECTORY	/END
	/F	DIRECTORY	/FAST
	/G	DIRECTORY	/START
	/J[:date]	DIRECTORY	/SINCE[date]
	/K[:date]	DIRECTORY	/BEGIN
	/L	*	
	/M	DIRECTORY	/FREE
	/N	DIRECTORY	/SUMMARY
	/O	DIRECTORY	/OCTAL
	/P	DIRECTORY	/EXCLUDE
	/Q	DIRECTORY	/DELETED
	/R	DIRECTORY	/REVERSE
	/S[:xxx]	DIRECTORY	/SORT[:category]
	/V[:ONL]	DIRECTORY	/VOLUMEID[:ONLY]
DUMP			
	/B	DUMP	/BYTES
	/E:n	DUMP	/END:block
	/G	DUMP	/IGNORE
	/N	DUMP	/NOASCII
	/O:n	DUMP	/ONLY:block
	/S:n	DUMP	/START:block
	/T	DUMP	/FOREIGN
	/W	DUMP	/WORDS
	/X	DUMP	/RAD50
DUP			
	/B[:RET]	INITIALIZE	/BADBLOCKS[:RET]
	/C	CREATE	
	/D	INITIALIZE	/RESTORE
	/E:n	COPY, CREATE	/END:n
	/F	COPY, DIRECTORY	/FILES
	/G:n	COPY, CREATE	/START:n
	/H	*	
	/I	COPY	/DEVICE
	/K	DIRECTORY	/BADBLOCKS
	/N:n	INITIALIZE	/SEGMENTS:n
	/O	BOOT	
	/Q	BOOT	/FOREIGN
	/R[:RET]	INITIALIZE	/REPLACE[:RETAIN]

(continued on next page)

Table B–2: System Monitor Command Equivalents (Cont.)

System Program	Option	Keyboard Monitor	
		Command	Option
DUP			
	/S	SQUEEZE	
	/T:n	CREATE	/EXTENSION:n
	/U[:xx]	COPY	/BOOT[:val]
	/V[:ONL]	INITIALIZE	/VOLUMEID[:ONLY]
	/W	INITIALIZE, COPY, SQUEEZE, BOOT	/WAIT
	/X	*	
	/Y	COPY, INITIALIZE, SQUEEZE	/NOQUERY
	/Z[:n]	INITIALIZE	
ERROUT			
	/A	SHOW ERRORS	/ALL
	/F:date	SHOW ERRORS	/FROM[:date]
	/S	SHOW ERRORS	/SUMMARY
	/T:date	SHOW ERRORS	/TO[:date]
FILEX			
	/A	COPY	/ASCII
	/D	DELETE	
	/F	DIRECTORY	/FAST
	/I	COPY	/IMAGE
	/L	DIRECTORY	
	/P	COPY	/PACKED
	/S	COPY	/DOS
	/T	COPY	/TOPS
	/U[:n.]	COPY	/INTERCHANGE[:size]
	/Y	INITIALIZE	/NOQUERY
	/Z	INITIALIZE	
FORMAT			
	/P:n	FORMAT	/PATTERN:n
	/S	FORMAT	/SINGLEDENSITY
	/V[:ONL]	FORMAT	/VERIFY[:ONLY]
	/W	FORMAT	/WAIT
	/Y	FORMAT	/NOQUERY
LIBR			
	/A	*	
	/C	LIBRARY	/PROMPT
	/D	LIBRARY	/DELETE
	/E	LIBRARY	/EXTRACT
	/G	LIBRARY	/REMOVE
	/N	*	
	/P	*	
	/R	LIBRARY	/REPLACE
	/U	LIBRARY	/UPDATE
	/W	*	
	/X	*	
	//	*	

(continued on next page)

Table B–2: System Monitor Command Equivalents (Cont.)

System Program	Option	Keyboard Monitor	
		Command	Option
LINK			
	/A	LINK	/ALPHABETIZE
	/B:n	LINK	/BOTTOM:n
	/C	LINK	/PROMPT
	/E:n	LINK	/EXTEND:n
	/F	*	
	/G	*	
	/H:n	LINK	/TOP
	/I	LINK	/INCLUDE
	/K:n	*	
	/L	LINK	/LDA
	/M[:n]	LINK	/STACK[:n]
	/O:n	*	
	/P:n	*	
	/Q	*	
	/R[:n]	LINK	/FOREGROUND[:STACKSIZE]
	/S	LINK	/SLOWLY
	/T[:n]	LINK	/TRANSFER[:n]
	/U:n	LINK	/ROUND:n
	/V:n[:m]	LINK	/XM
	/W	LINK	/WIDE
	/X	LINK	/NOBITMAP
	/Y:n	LINK	/BOUNDARY:value
	/Z:n	LINK	/FILL:n
		// *	
MACRO			
	/C:arg	MACRO	/CROSSREFERENCE [:type[...:type]]
	/D:arg	MACRO	/DISABLE[:value[...:value]]
	/E:arg	MACRO	/ENABLE[:value[...:value]]
	/L:arg	MACRO	/SHOW:value
	/M	MACRO	/LIBRARY
	/N:arg	MACRO	/NOSHOW:value
	/P:arg	MACRO	/PASS:1
ODT	Not accessible through keyboard monitor commands		
PAT	Not accessible through keyboard monitor commands		
PATCH	Not accessible through keyboard monitor commands		
PIP			
	/A	COPY	
	/B	COPY	/BINARY
	/C	COPY, DELETE, PRINT RENAME, TYPE	/NEWFILES
	/D	PRINT, TYPE	/DELETE
	/E	COPY, DELETE, PRINT, RENAME, TYPE	/WAIT
	/F	RENAME	/PROTECTION
	/G	COPY	/IGNORE
	/K:n	PRINT, TYPE	/COPIES:n
	/M:n	COPY, DELETE	/POSITION:n
	/N	COPY, RENAME	/NOREPLACE

(continued on next page)

Table B–2: System Monitor Command Equivalents (Cont.)

System Program	Option	Keyboard Monitor	
		Command	Option
PIP			
	/O	COPY	/PREDELETE
	/P	COPY, DELETE	/EXCLUDE
	/Q	COPY, DELETE, PRINT, RENAME, TYPE	/QUERY
	/R	RENAME	
	/S	COPY	/SLOWLY
	/T	COPY, RENAME	/SETDATE
	/U	COPY	/CONCATENATE
	/W	COPY, DELETE, PRINT, RENAME, TYPE	/LOG
	/Y	COPY, DELETE, RENAME	/SYSTEM
	/Z	RENAME	/NOPROTECTION
QUEMAN			
	/A	*	
	/D	PRINT	/DELETE
	/F	PRINT	/FLAGPAGE
	/H:n	PRINT	/FLAGPAGE:n
	/K:n	PRINT	/COPIES:n
	/L	SHOW	QUEUE
	/M	DELETE	/ENTRY
	/N	PRINT	/NOFLAGPAGE
	/P	*	
	/R	*	
	/S	*	
	//	PRINT	/PROMPT
RESORC			
	/A	SHOW	ALL
	/C	*	
	/D	SHOW	DEVICES
	/H	*	
	/J	SHOW	JOBS
	/L	SHOW	
	/M	*	
	/O	*	
	/T	SHOW	TERMINALS
	/Z	SHOW	CONFIGURATION
SIPP	Not accessible through keyboard monitor commands		
SLP	Not accessible through keyboard monitor commands		
SRCCOM			
	/A	DIFFERENCES	/AUDITTRAIL
	/B	DIFFERENCES	/BLANKLINES
	/C	DIFFERENCES	/NOCOMMENTS
	/D	DIFFERENCES	/CHANGEBAR
	/F	DIFFERENCES	/FORMFEED
	/L:n	DIFFERENCES	/MATCH:n
	/P	DIFFERENCES	/SLP
	/S	DIFFERENCES	/NOSPACES
	/T	DIFFERENCES	/NOTRIM
	/V:i:d	*	

A

/A option,
  DIR, 9-3
  LIBR, 12-3
  LINK, 11-40
  PIP, 7-11
  QUEMAN, 20-4
  RESORC, 17-2
  SRCCOM, 15-7
Abbreviating keyboard commands, 4-5
ABS,
  p-sect attribute, 11-4
. ABS., 11-3, 11-6
Absolute base address,
  specifying, 11-46
Absolute block parameters information (table), 11-15
Absolute section, 11-3, 11-6, See Also . ABS.
Access-code, 11-4
Accessing general registers,
  ODT, 21-9
Accessing internal registers,
  ODT, 21-10
Adding a subroutine to a module, 23-6
Address space,
  physical, 11-28
  program virtual (figure), 11-28
  virtual, 11-27
  virtual and physical (figure), 11-30, 11-32
Adjusting LINK library buffer size, 11-42
Advance command (A),
  EDIT, 5-20
ALL option,
  SHOW, 4-161
Alloc-code, 11-4, 11-5
/ALLOCATE option,
  COMPILE, 4-26
  COPY, 4-33
  CREATE, 4-44
  DIBOL, 4-52
  DIFFERENCES, 4-56
  DIRECTORY, 4-64
  DUMP, 4-74
  EDIT, 4-80
  EXECUTE, 4-83
  FORTRAN, 4-93
  LIBRARY, 4-114
/ALLOCATE option, (Cont)
  LINK, 4-120
  MACRO, 4-128
/ALPHABETIZE option,
  COMPILE, 4-26
  DIBOL, 4-52
  DIRECTORY, 4-64
  EXECUTE, 4-83
  LINK, 4-120
Altering or examining program contents, 21-1
/ALWAYS option,
  DIFFERENCES, 4-56
Analysis,
  error logging report, 19-6
APL keyboard command, 4-17
Arguments for /E and /D options (table), 10-7
Arguments for /L and /N options (table), 10-5
ASCII file format, 3-2
ASCII mode,
  copying files in, 7-11
/ASCII option,
  COPY, 4-33
  DUMP, 4-74
ASCII terminators (table), 21-20
ASCII text,
  changing while debugging, 21-20
ASCII values,
  displaying with SIPP, 22-7
ASECT directive, 11-16
.ASECT MACRO-11 directive, 11-3
Assembler,
  MAC8K, 10-13
  MACRO-11, 1-8
  using, 2-1
Assembling files, 4-82, 4-86, 4-128
Assembly language, See MACRO-11 assembly language
Assembly listing,
  sample (figure), 10-6
Assembly pass option, 10-13
ASSIGN keyboard command, 4-18
Assigning a job name for queued files, 4-135
Assigning a terminal to a foreground job, 4-100
Assigning a terminal to a system job, 4-170
Assigning logical names to devices, 4-18

Audit trail,
  specifying, 4-57, 15-7
/AUDITTRAIL option,
  DIFFERENCES, 4-57

B

B keyboard command, 4-20
/B option,
  DIR, 9-4
  DUP, 8-16
  LINK, 11-40
  PIP, 7-11
Back-arrow key,
  ODT, 21-8
Background job,
  directing control to, 3-7
  virtual, 4-137
Backing up through files,
  SIPP, 22-6
Backing up to a previous prompt,
  SIPP, 22-11
Bad blocks,
  covering, 4-108, 8-16
  eliminating, 4-90, 18-1
  finding locations of, 4-64, 8-7
  replacing, 4-108, 8-15
  scanning, 4-107, 8-6
Bad sector error, 4-35
/BADBLOCKS option,
  DIRECTORY, 4-64
  INITIALIZE, 4-107
Banner pages,
  printing for job, 4-134, 20-5
  setting default number of, 20-6
  suppressing printing of, 4-134, 20-6
Base,
  setting, 4-20
Base address,
  specifying absolute, 11-46
Base keyboard command, 4-20
Base-line monitor, See BL monitor
$BASIC command,
  BATCH, A-12
BASIC keyboard command, 4-21
BATCH, 1-8
  communicating with RT-11, A-35
  creating RT-11 mode programs, A-35
  differences between RT-11 and RSX-11D, A-50
  differences between RT-11 and RSX-11D (table), A-50
  hardware and software requirements, A-1
  loading, A-42
  RT-11 mode, A-34
  RT-11 mode examples, A-40
  running, A-44, A-1
BATCH character explanation (table), A-8
BATCH character set, A-8
BATCH command,
  $BASIC, A-12
  $CALL, A-13
  $CHAIN, A-14
  $COPY, A-14
  $CREATE, A-15
  $DATA, A-16
  $DELETE, A-18
  $DIRECTORY, A-18
  $DISMOUNT, A-18
  $EOD, A-19
  $EOJ, A-20
  $FORTRAN, A-20
  $JOB, A-22
  $LIBRARY, A-23
  $LINK, A-24
  $MACRO, A-26
  $MESSAGE, A-28
  $MOUNT, A-29
  $PRINT, A-31
  $RT11, A-31
  $RUN, A-31
  $SEQUENCE, A-32
BATCH command field options, A-3
BATCH command field options (table), A-3
BATCH command fields, A-2
BATCH command names, A-2
BATCH commands, A-11
BATCH commands (table), A-11
BATCH comment fields, A-7
BATCH control statement format, A-2
BATCH file specifications, A-6
BATCH file types (table), A-6
BATCH jobs,
  communicating with, A-47
BATCH operating procedures, A-42
BATCH programs,
  creating on punched cards, A-41
BATCH rules and conventions, A-10
BATCH specification field options, A-7
BATCH specification fields, A-5
BATCH streams,
  running SIPP from, 22-17
  sample, A-32
BATCH temporary files, A-9
BATCH wild card construction, A-6
/BEFORE option,
  DIRECTORY, 4-64
/BEGIN option,
  DIRECTORY, 4-64
Beginning command (B),
  EDIT, 5-20
Binary file comparison program, See BINCOM

Binary mode,
  copying files in, 4-33, 7-11
Binary number,
  format of 12-bit (figure),
    4-149
Binary object file format, 3-2
/BINARY option,
  COPY, 4-33
  DIFFERENCES, 4-57
BINCOM, 1-7, 16-1
  calling, 16-1
BINCOM examples, 16-4
BINCOM options, 16-2
BINCOM options (table), 16-2
BINCOM output format, 16-3
Bit map,
  inhibiting, 11-49
Bit patterns,
  selecting FORMAT verification,
    4-91, 18-3
/BITMAP option,
  LINK, 4-120
BL monitor, 1-3
  memory requirement, 1-3
Blank p-sect, 11-5, 11-6
Blank p-sect size, 11-6
/BLANKLINES option,
  DIFFERENCES, 4-58
/BLOCKS option,
  DIRECTORY, 4-65
Boot blocks
  copying, 4-34, 8-11
BOOT keyboard command , 4-22
/BOOT option,
  COPY, 4-34
Bootable devices,
  list of, 4-22, 8-8
Bootable volume,
  creating, 4-34, 8-8
Bootstrapping a device, 4-22, 8-8
Bootstrapping a different
    monitor, 4-22, 8-8
Bootstrapping a foreign volume,
    4-23, 8-9
Bootstrapping a non-RT-11 V04
    volume, 4-23, 8-9
/BOTTOM option,
  EXECUTE, 4-83
  LINK, 4-120
/BOUNDARY option,
  LINK, 4-121
Breakpoints,
  ODT, 21-11, 21-22
/BRIEF option,
  DIRECTORY, 4-65
BSE, See Bad sector error
/BUFFER option,
  FRUN, 4-99
  SRUN, 4-169
Bytes,
  advancing in with SIPP, 22-6
/BYTES option,
  DIFFERENCES, 4-58
  DUMP, 4-74

C

/C option,
  DIR, 9-4
  DUP, 8-3
  LIBR, 12-4
  LINK, 11-41
  PIP, 7-11
  RESORC, 17-2
/C option arguments (table),
    10-10
Cache memory error report,
  sample (figure), 19-8
Calculating offsets,
  ODT, 21-17
Calculating space for a
    foreground job, 4-99
$CALL command,
  BATCH, A-13
Callable subprograms,
  SYSLIB-FORTRAN, 1-11
Calling BINCOM, 16-1
Calling DIR, 9-1
Calling DUMP, 13-1
Calling DUP, 8-1
Calling EDIT, 5-1
Calling FILEX, 14-2
Calling FORMAT, 18-1
Calling LIBR, 12-1
Calling LINK, 11-7
Calling ODT, 21-1
Calling PAT, 23-1
Calling PATCH, 25-1
Calling PIP, 7-1
Calling QUEMAN, 20-2
Calling QUEUE, 20-1
Calling RESORC, 17-1
Calling SIPP, 22-1
Calling SLP, 24-1
Calling SRCCOM, 15-1
Calling the Error Logger, 19-3
Calling the MACRO-11 assemble,
    10-1
Calling the Queue Package, 20-1
Card,
  EOF (figure), A-42
Card reader device conditions,
  setting, 4-150
Cassette,
  using, 4-38, 4-50, 7-4
Cassette device conditions,
  setting, 4-150
Categories,
  sort (table), 4-69

$CHAIN command,
  BATCH, A-14
Change command (C),
  EDIT, 5-28
Change command arguments (table),
    5-29
/CHANGEBAR option,
  DIFFERENCES, 4-58
Changebars and bullets,
  inserting in a file comparison,
    4-58, 15-6
Changing amount of space
    allocated for library routine
    list, 11-46
Changing ASCII text while
    debugging, 21-20
Changing locations,
  ODT, 21-7
  PATCH, 25-4
Changing monitors, 4-22, 8-8
Changing the number of directory
    segments, 4-110, 8-14
Changing volume identification,
    8-12, 8-15
Changing volumes during an
    operation, 4-23, 4-41, 4-51,
    4-72, 4-92, 4-110, 4-136,
    4-142, 7-16, 8-13
Character explantion,
  BATCH (table), A-8
Character set,
  BATCH, A-8
Characters,
  prompting (table), 6-3
Checksum,
  PATCH, 25-2
  SIPP, 22-15
Circumflex key,
  ODT, 21-8
Clearing a device directory,
    4-107, 8-14
CLOSE keyboard command, 4-24
Closing and opening files, 5-12
Closing locations,
  ODT, 21-7
Closing output files, 4-24
Code modifications,
  completing with SIPP, 22-12
/CODE option,
  COMPILE, 4-26
  EXECUTE, 4-83
  FORTRAN, 4-94
Codes,
  FORTRAN listing (table), 4-96
  FORTRAN optimization (table),
    4-96
/COLUMNS option,
  DIRECTORY, 4-65
Combining library option
    functions, 12-12
Combining low memory and extended
    memory overlays, 11-34
Command,
  Edit Backup, 5-13
  Edit Read, 5-12
  Edit Write, 5-13
  End File, 5-14
  syntax of keyboard, 4-1
Command string,
  continuing (QUEMAN), 20-10
Command arguments,
  EDIT, 5-5
  EDIT change (table), 5-29
  list (table), 5-24
  write (table), 5-17
Command categories,
  EDIT (table), 5-4
Command fields,
  BATCH, A-2
Command file,
  creating a SLP, 24-4
Command file operators,
  SLP (table), 24-4
Command files,
    See Indirect command files
  indirect, 4-9
Command names,
  BATCH, A-2
Command properties,
  EDIT, 5-7
Command repitition,
  EDIT, 5-9
Command string,
  continuing (LIBR), 12-4
  continuing (LINK), 11-41
  EDIT, 5-6
Command string interpreter,
    See CSI
Command structure,
  EDIT, 5-3
Commands,
    See Also Keyboard commands
  BATCH, A-11
  BATCH (table), A-11
  EDIT editing, 5-12
  EDIT utility, 5-30
  immediate mode (table), 5-37
  ODT single-instruction mode
    (table), 21-14
  PATCH, 25-2
  PATCH (table), 25-3
  SIPP, 22-4
  SIPP (table), 22-4
Commands and functions,
  ODT, 21-6
Commands supporting wildcards
    (table), 4-8
Comment fields,
  BATCH, A-7

/COMMENTS option,
  DIFFERENCES, 4-58
COMMON,
  FORTRAN global section, 11-5
Communicating with BATCH jobs,
    A-47
Communicating with RT-11,
  BATCH, A-35
Communication,
  system, II-1
Communication links,
  between modules, 11-6
Comparing two files, 4-56, 15-1,
    16-1
Comparison program,
  binary, See BINCOM
  source, See SRCCOM
COMPILE keyboard command, 4-25
Compiling files, 4-25, 4-82
Compiling FORTRAN IV programs,
    4-93
Complete listing,
  system resources, 4-161, 17-2
Completing code modifications,
  SIPP, 22-12
Complex keyboard commands, 1-9
Components,
  hardware (table), 1-2
  system hardware, 1-1
  system software, 1-2
Compressing a device, 4-167,
    8-9
CON,
  p-sect attribute, 11-4
/CONCATENATE option,
  COPY, 4-35
Concatenating files, 4-35, 7-13
CONFIGURATION option,
  SHOW, 4-161
Configuration parameters,
  setting system, 4-149
Confirmation message,
  suppressing, 4-40, 4-51, 4-91,
    4-108, 4-168, 8-13, 18-5
Constant register,
  ODT, 21-16
Contents of queue,
  listing, 4-165, 20-5
Continuing a command string,
  (LIBR), 12-4
  (QUEMAN), 20-10
  (LINK), 11-41
Control characters,
  PATCH (table), 25-5
Control commands,
    See CTRL commands
Control statement format,
  BATCH, A-2
Controlling program execution
    while debugging, 21-12
Conventions,
  system, 3-1
Conventions and rules,
  BATCH, A-10
Copies,
  generating copies of files,
    4-133, 7-12
  printing multiple, 20-5
/COPIES option,
  PRINT, 4-133
  TYPE, 4-174
$COPY command,
  BATCH, A-14
COPY keyboard command, 4-32
Copying a device, 4-35, 8-5
Copying a device to a file, 8-5
Copying a file to a device, 8-5
Copying boot blocks, 4-34, 8-11
Copying devices, 8-5
Copying files, 4-32, 7-1
  block by block, 4-41
  block-by-block, 7-13
  entering current date while,
    4-41, 7-13
  except those specified, 7-13
  except those specified , 4-36
  in ASCII mode, 7-11
  in binary mode, 4-33, 7-11
  in image mode, 7-10
  of the current date, 4-37, 7-11
  system, 4-41, 7-13
  to magtape or cassette, 4-38
  with a diskette system, 4-42,
    7-17
  with a single volume system,
    4-42, 7-16
Copying system files, 4-41, 7-13
Copying volumes, 8-5
Correction file,
  PAT, 23-4
Covering bad blocks, 4-108, 8-16
$CREATE command,
  BATCH, A-15
CREATE keyboard command, 4-44
/CREATE option,
  EDIT, 4-80
  LIBRARY, 4-114
Creating a bootable volume, 4-34,
    8-11
Creating a Macro library, 4-115,
    12-14
Creating a patch command file,
    4-56, 15-7, 16-5, 22-1
Creating a patch for a memory
    image file, 4-56, 16-5
Creating a patch for a source
    file, 4-56, 15-7, 24-4
Creating a scratch region in
    extended memory, 4-125, 11-49
Creating a SLP command file, 15-7

Creating a system volume, 8-11
Creating an LDA file, 11-44
Creating an overlay structure, 11-18
Creating BATCH programs on punched cards, A-41
Creating files, 4-79, 8-3
Creating indirect command files, 4-9
Creating library files, 4-113, 12-4
Creating multiple definition libraries, 12-10
Creating RT-11 mode BATCH programs, A-35
Cross-reference sections (table), 4-129
Cross-reference table,
  obtaining a, 10-9
Cross-reference table (figure), 10-12
Cross-reference table files,
  handling, 10-10
Cross-reference table generation option, 10-9
/CROSSREFERENCE option,
  COMPILE, 4-26
  DIBOL, 4-52
  EXECUTE, 4-84
  MACRO, 4-128
.CSECT MACRO-11 directive, 11-4, 11-5
CSI, 6-1
CSI prompting characters, 6-3
CSI syntax, 6-1
CTRL commands, 3-6
CTRL/A, 3-7
CTRL/B, 3-7
CTRL/C, 3-7, 5-3
CTRL/E, 3-7
CTRL/F, 3-7
CTRL/O, 3-7, 5-3
CTRL/Q, 3-8
CTRL/S, 3-8
CTRL/U, 3-8, 5-3
CTRL/X, 3-8, 5-3
CTRL/Y, 22-12
CTRL/Z, 3-8, 22-11
Current location pointer, 5-7
Current monitor,
  resource listing, 4-161, 17-6
Cycle,
  program development (figure), 2-3

D

D,
  p-sect attribute, 11-4
D keyboard command, 4-46
/D option,
  DIR, 9-4
  DUP, 8-16
  LIBR, 12-6
  PIP, 7-14
  QUEMAN, 20-4
  RESORC, 17-3
  SRCCOM, 15-6
$DATA command,
  BATCH, A-16
Data formats, 3-2
Data p-sect, 11-5
Date,
  inspecting system, 4-47
  setting system, 4-47
DATE keyboard command, 4-47
/DATE option,
  DIRECTORY, 4-66
DEASSIGN keyboard command, 4-48
/DEBUG option,
  EXECUTE, 4-84
  LINK, 4-121
Debugger,
  using, 2-2
Debugging technique,
  on-line, See ODT
DECsystem-10,
  transferring files to RT-11 from, 14-8
DECtape II device conditions,
  setting, 4-150
DECtapes,
  deleting files from DOS/BATCH, 14-10
Default directory sizes (table), 4-110, 8-15
Default file specification values (table), 10-3
Default formatting, 4-90, 18-3
Defaults,
  linker (table), 11-8
  wildcard (table), 4-8
$DELETE command,
  BATCH, A-18
Delete command (D),
  EDIT, 5-26
DELETE key, 3-8
DELETE keyboard command, 4-49
/DELETE option,
  LIBRARY, 4-114
  PRINT, 4-133
  TYPE, 4-174
Deleted files,
  recovering, 4-44, 8-4
/DELETED option,
  DIRECTORY, 4-66
Deleting current input line, 3-8
Deleting files, 4-49, 7-14
  from DOS/BATCH DECtapes, 14-10
  from interchange diskettes, 14-10

Deleting global symbols from a
library directory, 12-7
Deleting input files after
printing, 20-5
Deleting modules from a library,
12-6
Deleting QUFILE.TMP, 20-6
Deleting system files, 4-51
Deletion,
enabling files for, 4-141, 7-15
protecting files from, 4-141,
7-15
Deposit keyboard command, 4-46
Depositing a value, 4-46
Description,
sample load map (table), 11-17
Description of sample load map
(table), 11-37
Development cycle,
program (figure), 2-3
Device, See Also Volume
Device assignments,
resource listing, 4-163, 17-5
Device directory,
See Also Directory
clearing, 4-107
Device handler status,
resource listing, 4-161, 17-3
Device handlers, 1-5
loading, 4-126
unloading, 4-176
Device identification,
displaying or changing, 4-72,
8-12
Device names,
permanent (table), 3-3
physical (table), 3-3
/DEVICE option,
COPY, 4-35
Device structures, 3-5
Device structures (table), 3-6
Device utility program, See DUP
Devices, See Also Volume
bootstrapping, 4-22, 8-8
compressing, 4-167, 8-9
copying, 4-35, 8-5
creating bootable, 4-34, 8-8,
8-11
directory-structured, 3-5
dumping, 4-73, 13-1
examining and modifying, 22-1
file-structured, 3-5
FILEX supported (table), 14-1
formatting, 4-90, 18-1
installing, 4-112
list of bootable, 4-22, 8-8
maintaining, 8-1
random-access, 3-5
sequential-access, 3-5
Devices, (Cont)
SET conditions and
modifications (table), 4-150
squeezing, 4-167, 8-9
storage error report, 19-6
supported by FILEX, 14-1
DEVICES option,
SHOW, 4-163
/DIAGNOSE option,
COMPILE, 4-27
EXECUTE, 4-84
FORTRAN, 4-94
Dialogue,
SIPP, 22-3
DIBOL keyboard command, 4-52
/DIBOL option,
COMPILE, 4-27
EXECUTE, 4-84
Differences between RT-11 BATCH
and RSX-11D BATCH, A-50
Differences between RT-11 BATCH
and RSX-11D BATCH (table),
A-50
DIFFERENCES keyboard command,
4-56
DIR, 1-6, 9-1
calling, 9-1
DIR options, 9-2
DIR options (table), 9-2
Directing control to background
job, 3-7
Directing control to foreground
job, 3-7
Directing control to system job,
3-8
Directive summary,
.DSABL and .ENABL (table),
4-129
Directives,
BATCH run-time handler operator
(table), A-48
Directory,
default sizes (table), 8-15
$DIRECTORY command,
BATCH, A-18
DIRECTORY keyboard command, 4-62
Directory listing,
a summary, 9-7
all but files specified, 9-8
alphabetically, 4-64, 9-3
beginning with specified file,
4-64, 9-5
block numbers in octal, 9-7
by category, 4-68, 9-9
deleted files, 4-66, 9-8
files created after specified
date, 4-70, 9-6
files created before specified
date, 4-64, 9-6

Directory listing, (Cont)
  files of a specified date, 4-66, 9-4
  files' starting block numbers, 4-65, 9-4
  FILEX, 14-9
  in reverse, 4-70, 9-9
  library file, 12-10
  obtaining, 4-62
  on a line printer, 4-70
  on line printer, 9-6
  only file names and types, 4-65, 9-5
  setting number of columns in listing, 4-65, 9-4
  unused areas, 4-67, 9-7
  used and unused areas, 4-67, 9-5
  volume identification, 4-72, 9-10
Directory program, See DIR
Directory segments,
  changing the number of, 4-110, 8-14
Directory sizes,
  default (table), 4-110
Directory-structured volumes, 3-5
/DISABLE option,
  COMPILE, 4-27
  EXECUTE, 4-84
  MACRO, 4-129
Diskette,
  formatting in single-density mode, 4-91, 18-5
Diskette system,
  copying files with, 4-41, 7-17
$DISMOUNT command,
  BATCH, A-18
Display editor,
  EDIT, 5-34
  using, 5-35
Display editor format,
  12-inch screen (figure), 5-35
Display hardware,
  using, 4-103
Display screen values (table), 4-109
Displaying ASCII values with SIPP, 22-7
Displaying Radix-50 values with SIPP, 22-8
Displaying system time, 4-173
Displaying volume identification, 4-72, 8-12
Documentation,
  RT-11 software, 1-9
/DOS option,
  COPY, 4-36
  DELETE, 4-49
  DIRECTORY, 4-66
  INITIALIZE, 4-108
DOS/BATCH and RT-11,
  transferring files between, 14-3
DOS/BATCH DECtapes,
  deleting files from, 14-10
.DSABL and .ENABL MACRO-11 directive summary (table), 4-129
Dummy file name, 8-4
DUMP, 1-6, 13-1
  calling, 13-1
DUMP examples, 13-2
DUMP keyboard command, 4-73
DUMP options, 13-1
DUMP options (table), 13-1
Dump program, See DUMP
Dumping devices and files, 4-73
DUP, 1-6, 8-1
  calling, 8-1
DUP options, 8-1
DUP options (table), 8-2
DUP options and categories (table), 8-2

E

E keyboard command, 4-78
/E option,
  DIR, 9-5
  LIBR, 12-6
  LINK, 11-41
  PIP, 7-16
EDIT, 1-5, 4-79, 5-1
  calling, 5-1
  modes of operation, 5-1
  Edit Backup command, 5-13
  EDIT command arguments, 5-5
  EDIT command categories (table), 5-4
  EDIT command properties, 5-7
  EDIT command repitition, 5-9
  EDIT command string, 5-6
  EDIT command structure, 5-3
  EDIT commands and file status (table), 5-15
  EDIT display editor, 5-34
  EDIT editing commands, See Editing commands
  EDIT error conditions, 5-39
  EDIT example, 5-38
  EDIT key commands (table), 5-2
  EDIT keyboard command, 4-79
  Edit Lower command (EL), EDIT, 5-34
  EDIT memory usage, 5-10
  Edit Read command, 5-12
  Edit Upper command (EU), EDIT, 5-34
Edit Version command (EV), EDIT, 5-33

Edit Write command, 5-13
Editing text files, 4-79
Editing commands,
  EDIT, 5-12
Editing text files, 1-5
Editor,
  EDIT, 1-5, 4-79, 4-151, 5-1
  K52, 1-5, 4-79, 4-81, 4-151
  KED, 1-5, 4-79, 4-81, 4-151
  keypad, See KED or K52
  setting default, 4-151
  TECO, 4-151
  using, 2-1
Effective address search,
  ODT, 21-15
EL, 19-2
EL.REL, 19-2
EL.SYS, 19-2
Eliminating bad blocks, 4-90,
    18-1
ELINIT, 19-2
  using, 19-4
.ENABL GBL MACRO-11 directive,
    11-6
/ENABLE option,
  COMPILE, 4-27
  EXECUTE, 4-84
  MACRO, 4-129
Enabling files for deletion,
    4-141, 7-15
Enabling special .SETTOP and
    .LIMIT features, 4-125, 11-49
End File command, 5-14
/END option,
  COPY, 4-36
  DIFFERENCES, 4-58
  DIRECTORY, 4-67
  DUMP, 4-74
Entering ASCII values,
  SIPP, 22-7
Entering octal values,
  SIPP, 22-7
Entering Radix-50 values,
  SIPP, 22-8
/ENTRY option,
  DELETE, 4-49
Entry point, 11-6
$EOD command,
  BATCH, A-19
EOF card (figure), A-42
$EOJ command,
  BATCH, A-20
ERRLOG.DAT, 19-2
Error,
  bad sector, 4-35
Error abort level,
  setting indirect command file,
    4-151
Error codes,
  MACRO-11, 10-14
  MACRO-11 (table), 10-14
Error conditions,
  EDIT, 5-39
Error detection,
  ODT, 21-26
Error Logger, 1-7, 19-1
  calling, 19-3
Error Logging, 19-1
Error logging report analysis,
    19-6
Error Logging subsystem, 19-2
Error logging subsystem (figure),
    19-3
Error logging uses, 19-1
Error report,
  analysis of memory parity
    (table), 19-8
  analysis of sample storage
    device (table), 19-7
  memory, 19-8
  sample cache memory (figure),
    19-8
  sample memory parity (figure),
    19-8
  sample report file environment
    and error count (figure),
    19-10
  sample storage device (figure),
    19-7
  sample summary for device
    statistics (figure), 19-9
  sample summary for memory
    statistics (figure), 19-10
  storage device, 19-6
  Summary, 19-9
Errors during file copy,
  ignoring input, 7-12
ERRORS option,
  SHOW, 4-164
ERROUT, 19-2
  using, 19-5
.EVEN MACRO-11 directive, 11-5
Examining and modifying devices,
    22-1
Examining and modifying save
    image files, See SIPP
Examining locations in a file,
  PATCH, 25-4
Examining memory locations, 4-78
Examining or altering program
    contents, 21-1
Examples,
  BATCH RT-11 mode, A-40
  BINCOM, 16-4
  creating an extended memory
    overlay structure, 11-31
  DUMP, 13-2
  PATCH, 25-8
  SLP, 24-3
  specifying low memory overlay
    structure, 11-44
  updating object modules, 23-5

Exchange command (X),
  EDIT, 5-30
Exchange command arguments,
  EDIT (table), 5-30
/EXCLUDE option,
  COPY, 4-36
  DELETE, 4-50
  DIRECTORY, 4-67
EXECUTE keyboard command, 4-82
Execute macro command (EM),
  EDIT, 5-33
/EXECUTE option,
  EDIT, 4-81
  EXECUTE, 4-84
  LINK, 4-121
Executing a system job, 4-169
Executing background virtual
    jobs, 4-137
Executing foreground jobs, 4-99
Executing indirect command files,
    4-12
Executing virtual .SAV files,
    4-99
Executing virtual foreground
    jobs, 4-99
Exit command (EX),
  EDIT, 5-18
Exiting from PATCH, 25-4
/EXTEND option,
  COMPILE, 4-27
  EXECUTE, 4-85
  FORTRAN, 4-94
  LINK, 4-121
Extended memory, 11-2
  creating a scratch region in,
    4-125, 11-49
Extended memory and low memory
    overlays,
  memory diagram showing
    (figure), 11-35
Extended memory monitor,
    See XM monitor
Extended memory overlay handler
    (figure), 11-38
Extended memory overlay option,
    11-31
Extended memory overlay
    structure,
  examples, 11-31
Extended memory overlays, 11-27
  load map for program with
    (figure), 11-36
Extended memory partitions that
    contain sharing segments
    (figure), 11-33
Extending a file, 4-44, 8-10
Extending files and overlay
    segments with SIPP, 22-12
Extending p-sect in root, 11-41
/EXTENSION option,
  CREATE, 4-44
/EXTRACT option,
  LIBRARY, 4-114
Extracting modules from a
    library, 4-114, 12-6

F

/F option,
  DIR, 9-5
  DUP, 8-7
  LINK, 11-42
  PIP, 7-15
Factoring file specifications,
    4-3
/FAST option,
  DIRECTORY, 4-67
FB monitor, 1-3
  memory requirement, 1-3
Features,
  special,
    See Also Special features
File copy operations, 7-1,
    See Copying files
File exchange program, See FILEX
File format,
  .REL image, 3-2
  .SAV image, 3-2
  ASCII, 3-2
  binary object, 3-2
  LDA image, 3-2
File formats, 14-1
File name,
  dummy, 8-5
File names, 3-4
/FILE option,
  INITIALIZE, 4-108
File protection, 4-141, 7-15
File specification syntax, 6-1
File specification values,
  default (table), 10-3
File specifications,
  BATCH, A-6
  factoring, 4-3
  keyboard command, 4-3
File specifications options,
  MACRO (table), 10-4
File transfer operations,
    See Also Copying files
File transfers involving magtape
    and cassette, 4-38, 7-4
File type specifications, 4-5
File types, 3-4
  BATCH (table), A-6
  standard (table), 3-4
File-structured devices, 3-5
Files,
  assembling, 4-82, 4-128, 10-1

Files, (Cont)
assigning a job name for queued, 4-135
comparing two, 4-56, 15-1, 16-1
compiling, 4-25, 4-82
concatenating, 4-35, 7-13
copying, 4-32, 7-1, 7-10
copying of the current date, 4-37, 7-11
copying or transferring, 4-35, 7-1
copying system, 4-41, 7-13
copying with a diskette system, 4-42, 7-17
copying with a single volume system, 4-42, 7-16
creating, 4-79, 8-4
creating a patch for memory image, 16-5
deleting, 4-49, 7-14
deleting from DOS/BATCH DECtapes, 14-10
deleting from interchange diskettes, 14-10
deleting input after printing, 20-5
deleting system, 4-51
dumping, 4-73, 13-1
editing, 4-79
enabling for deletion, 4-141, 7-15
extending, 4-44, 8-10
extending with SIPP, 22-12
generating copies of, 4-133, 7-12
ignoring input errors during copy, 4-36, 7-12
indirect command, 4-9
load map, 4-119
loading into memory, 4-102
logging, 7-15
oerating on system, 4-142
opening and closing, 5-12
operating on system, 4-41, 4-51, 7-13
PAT input, 23-4
patching save image, See SIPP
patching source, See SLP
preventing replace during copy, 4-40, 7-12
printing, 4-133
printing multiple copies, 20-5
protecting from deletion, 4-141, 7-15
queuing for subsequent printing, 4-133, 20-1
recovering deleted, 4-44, 8-4
renaming, 4-140, 7-14
renaming system, 4-142
Files, (Cont)
searching through with SIPP, 22-9
symbol definition table, 4-119
transferring, 4-32, 7-1
transferring between RT-11 and DOS/BATCH, 14-3
transferring between RT-11 and interchange diskette, 14-5
transferring to RT-11 from DECsystem-10, 14-8
typing at terminal, 4-174
/FILES option,
COPY, 4-36
DIRECTORY, 4-67
FILEX, 1-7, 14-1
calling, 14-2
FILEX options, 14-2
FILEX options (table), 14-3
FILEX supported devices, 14-1
FILEX supported devices (table), 14-1
/FILL option,
LINK, 4-122
Filling unused locations in load module, 11-50
Find command (F),
EDIT, 5-23
Finding locations of bad blocks, 8-7
/FLAGPAGE option,
PRINT, 4-134
Foreground jobs,
assigning a terminal to 4-100
calculating space for, 4-99
directing control to, 3-7
initiating, 4-99
linking, 4-122, 11-47
unloading, 4-176
/FOREGROUND option,
LINK, 4-122
Foreground/background jobs,
using ODT with, 21-21
Foreground/background monitor, See FB monitor
Foreground/background terminal I/O, 3-8
/FOREIGN option,
BOOT, 4-23
DUMP, 4-74
FORMAT, 1-7, 18-1
calling, 18-1
specifying verification bit patterns, 4-91, 18-3
Format,
BATCH control statement, A-2
BINCOM output, 16-3
ODT printout, 21-6
SRCCOM output, 15-3

FORMAT keyboard command, 4-90
Format of a 12-bit binary number (figure), 4-149
FORMAT options, 18-2
FORMAT options (table), 18-2
Formats,
  data, 3-2
Formatting,
  default, 18-3
Formatting a diskette in single-density mode, 4-91, 18-5
Formatting volumes, 4-90, 18-1
/FORMFEED option,
  DIFFERENCES, 4-58
Forms of ODT relocatable expressions (r) (table), 21-5
$FORTRAN command,
  BATCH, A-20
FORTRAN IV,
  library modules, 11-11
FORTRAN keyboard command, 4-93
FORTRAN library modules,
  linking, 11-42
FORTRAN listing codes (table), 4-96
FORTRAN optimization codes (table), 4-96
/FORTRAN option,
  COMPILE, 4-27
  EXECUTE, 4-85
FORTRAN programs,
  p-sect ordering for, 11-6
/FREE option,
  DIRECTORY, 4-67
FRUN keyboard command, 4-99
/FULL option,
  DIRECTORY, 4-67
Function control options,
  MACRO-11, 10-7

G

/G option,
  DIR, 9-5
  LIBR, 12-7
  LINK, 11-42
  PIP, 7-12
GBL,
  p-sect attribute, 11-4
General registers,
  accessing with ODT, 21-9
Generating copies of files, 4-133, 7-12
Get command (G),
  EDIT, 5-21
GET keyboard command, 4-102
Global references,
  resolving, 11-7
Global symbol, 11-2
Global symbols, 11-6
  deleting from a library directory, 12-7
  including all in library's directory, 12-3
  including from any library in link, 11-43
  listing in alphabetical order, 11-40
  unresolved, 4-119
.GLOBL MACRO-11 directive, 11-6
GT keyboard command, 4-103
GT OFF keyboard command, 4-103
GT ON keyboard command, 4-103
Guidelines for specifying an overlay structure, 11-23

H

/H option,
  LINK, 11-42
  QUEMAN, 20-4
  RESORC, 17-4
Halting QUEUE, 20-4
Handler,
  device, See device handlers
  extended memory overlay (figure), 11-38
Handling cross-reference table files, 10-10
Hardware and software requirements,
  BATCH, A-1
Hardware bootstrap, 4-22
  performing, 8-8
Hardware components (table), 1-2
Hardware configuration,
  resource listing, 17-4
Hardware vectors, 11-3
/HEADER option,
  COMPILE, 4-27
  EXECUTE, 4-85
  FORTRAN, 4-94
HELP keyboard command, 4-105
Highest address used by relocatable code,
  specifying, 11-42

I

I,
  p-sect attribute, 11-4
/I option,
  DUP, 8-4
  LINK, 11-43
I/O,
  foreground/background terminal, 3-8
/I4 option,
  COMPILE, 4-27

/I4 option, (Cont)
EXECUTE, 4-85
FORTRAN, 4-94
/IGNORE option,
COPY, 4-36
DUMP, 4-74
Ignoring input errors during file copy, 4-36, 7-12
Image mode,
copying files in, 4-37, 7-10
/IMAGE option,
COPY, 4-37
Immediate mode, 5-37
Immediate mode commands (table), 5-37
/INCLUDE option,
LINK, 4-122
Including all global symbols in library's directory, 12-3
Including global symbols from any library in link, 11-43
Including module names in a library directory, 12-7
Including p-sect names in library directory, 12-8
Indirect command file,
running SIPP from, 22-16
Indirect command file error abort level,
setting, 4-151
Indirect command files, 4-9
creating, 4-9, 15-7, 16-5
executing, 4-12
interacting with, 4-10
nesting, 4-14
SIPP, 22-16
SLP, 24-4
startup, 3-2, 4-15
Information,
HELP keyboard command, 4-105
Inhibiting a bit map, 11-49
INITIALIZE keyboard command, 4-107
Initializing a directory, 4-107, 8-14
Initiating a foreground job, 4-99
Input and output,
LINK, 11-11
Insert command (I),
EDIT, 5-25
/INSERT option,
LIBRARY, 4-115
Inserting changebars and bullets in a file comparison, 4-58, 15-6
Inserting modules into a library, 4-115, 7-5
Inserting value into word 56 of .SAV file, 11-43
/INSPECT option,
EDIT, 4-81
Inspecting system date, 4-47
INSTALL keyboard command, 4-112
Installing devices, 4-112
Interacting with indirect command files, 4-10
Interactive commands,
See keyboard commands
Interchange diskette and RT-11,
transferring files between, 14-5
Interchange diskettes,
deleting files from, 4-50, 14-10
/INTERCHANGE option,
COPY, 4-37
DELETE, 4-50
DIRECTORY, 4-68
INITIALIZE, 4-108
Internal ODT registers (table), 21-10
Internal registers,
accessing with ODT, 21-10
Invoking a language processor, 4-25, 4-82

J

/J option,
DIR, 9-6
RESORC, 17-5
Job,
privileged, 1-5, 11-31
virtual, 1-5, 11-31
$JOB command,
BATCH, A-22
Jobs,
assigning a name to, 4-135
deleting from queue, 4-49, 20-5
loaded,
resource listing, 17-5
printing banner pages, 20-5
resuming or restarting printing of, 20-7
suspending printing of, 20-7
system, 1-7
JOBS option,
SHOW, 4-163
Jump command (J),
EDIT, 5-20

K

/K option,
DIR, 9-6
DUP, 8-6
LINK, 11-43
PIP, 7-12
QUEMAN, 20-5

K52, 1-5, 4-79, 4-151
/K52 option,
  EDIT, 4-81
KED, 1-5, 4-79, 4-151
/KED option,
  EDIT, 4-81
Key commands,
  EDIT, 5-2
  EDIT (table), 5-2
Keyboard command,
  APL, 4-17
  ASSIGN, 4-18
  B, 4-20
  Base, 4-20
  BASIC, 4-21
  BOOT, 4-22
  CLOSE, 4-24
  COMPILE, 4-25
  COPY, 4-32
  CREATE, 4-44
  D, 4-46
  DATE, 4-47
  DEASSIGN, 4-48
  DELETE, 4-49
  Deposit, 4-46
  DIBOL, 4-52
  DIFFERENCES, 4-56
  DIRECTORY, 4-62
  DUMP, 4-73
  E, 4-78
  EDIT, 4-79
  EXECUTE, 4-82
  FORMAT, 4-90
  FORTRAN, 4-93
  FRUN, 4-99
  GET, 4-102
  GT, 4-103
  HELP, 4-105
  INITIALIZE, 4-107
  INSTALL, 4-112
  LIBRARY, 4-113
  LINK, 4-119
  LOAD, 4-126
  MACRO, 4-128
  PRINT, 4-133
  R, 4-137
  REENTER, 4-138
  REMOVE, 4-139
  RENAME, 4-140
  RESET, 4-143
  RESUME, 4-144
  RUN, 4-145
  SAVE, 4-147
  SET, 4-149
  SHOW, 4-160
  SQUEEZE, 4-167
  SRUN, 4-169
  START, 4-171
  SUSPEND, 4-172
  TIME, 4-173
Keyboard command, (Cont)
  TYPE, 4-174
  UNLOAD, 4-176
Keyboard command file
    specifications, 4-3
Keyboard command mutually
    exclusive options, 4-2
Keyboard command option, 4-1
Keyboard command prompt, 4-6
Keyboard command prompts, 4-1
Keyboard command syntax, 4-1
Keyboard command/system program
    equivalents (table), B-1
Keyboard commands, 4-1, 4-15
  abbreviating, 4-5
  complex, 1-9
  removing, 4-15
  simple, 1-9
  summary, B-1
  using, 1-9
Keyboard monitor, See KMON
Keyboard monitor commands,
    See Keyboard commands
Keyboard monitor prompt, 4-1
Keypad editor, See KED or K52
Keys,
  special function, 3-6
Kill command (K),
  EDIT, 5-27
KMON, 1-2

L

/L option,
  DIR, 9-6
  GT, 4-103
  LINK, 11-44
  QUEMAN, 20-5
  RESORC, 17-5
Language,
  assembly, 1-8
Language processors, 1-8
  invoking, 4-25, 4-82
LCL,
  p-sect attribute, 11-4
LDA file,
  producing, 11-44
LDA file format, 3-2
LDA image file, 3-2
/LDA option,
  LINK, 4-122
/LEVEL option,
  SRUN, 4-169
LIBR, 1-6, 12-1
  calling, 12-1
  using, 2-2
LIBR Macro options (table), 12-13
LIBR object options (table), 12-3
Librarian program, See LIBR

Library,
  creating multiple definition, 12-10
  deleting global symbols from directory, 12-7
  deleting modules from, 4-114, 12-6
  extracting modules from, 4-114, 12-6
  including module names in directory, 12-7
  including p-sect names in directory, 12-8
  inserting modules into, 4-115, 7-5
  listing directory, 4-115, 12-10
  merging files, 12-11
  replacing modules in, 12-8, 4-117
  searching, 4-119
  updating, 4-117, 12-9
  using multiple definition, 11-14
Library buffer size,
  adjusting LINK, 11-42
$LIBRARY command,
  BATCH, A-23
Library files, 11-2
  creating, 4-113, 12-4
LIBRARY keyboard command, 4-113
Library modules, 11-2, 11-11
  linking FORTRAN, 11-42
  processing, 11-12
/LIBRARY option,
  COMPILE, 4-27
  EXECUTE, 4-85
  LINK, 4-122
  MACRO, 4-129
LIBRARY options,
  prompting sequence (table), 4-117
Library routine list,
  changing amount of space allocated for, 11-46
Library searches (figure), 11-13
.LIMIT programmed request, 4-125
Limits,
  overlaid program segment (table), 22-14, 22-15
LINE FEED key,
  ODT, 21-8
Line printer device conditions,
  setting, 4-152
Line-by-line analysis of the sample memory error report (table), 19-8
Line-by-line analysis of the sample storage device error report (table), 19-7
Line-by-line sample load map description (table), 11-17, 11-37
/LINENUMBERS option.
  COMPILE, 4-27
  DIBOL, 4-53
  EXECUTE, 4-85
  FORTRAN, 4-94
Lines in a module,
  overlaying,
  PAT, 23-5
Lines of input to LINK,
  typing additional, 11-41
LINK, 1-6, 11-1
  calling, 11-7
  memory requirement for, 11-3
$LINK command,
  BATCH, A-24
LINK input and output, 11-11
LINK keyboard command, 4-119
LINK options prompting sequence (table), 4-120
LINK prompts, 11-50
Linker, See Also LINK
  using, 2-2
Linker defaults (table), 11-8
Linker functions, 11-2
Linker options (table), 11-9
Linker process,
  overview, 11-1
Linker program, See LINK
Linking a foreground job, 4-122, 11-47
Linking FORTRAN library modules, 11-42
Linking object modules, 4-119, 11-1
/LINKLIBRARY option,
  EXECUTE, 4-85
  LINK, 4-122
.LIST and .NLIST MACRO-11 directive summary (table), 4-132
List command (L),
  EDIT, 5-24
List command arguments (table), 5-24
/LIST option,
  COMPILE, 4-28
  DIBOL, 4-53
  EXECUTE, 4-85
  FORTRAN, 4-94
  LIBRARY, 4-115
  MACRO, 4-130
Listing,
  requesting a wide (LIBR), 12-9
  system resources,
    See Resource listing
Listing a device directory, 4-62, 9-1

Listing bad blocks, 4-64, 8-6
Listing codes,
  FORTRAN (table), 4-96
Listing contents of the queue,
    4-165, 20-5
Listing control options,
  MACRO-11, 10-4
Listing differences between two
    files, 4-56, 15-1, 16-1
Listing directories (FILEX), 14-9
Listing global symbols in
    alphabetical order, 4-120,
    11-40
Listing volume directories,
    See Directory listing
Load image file format, 3-2
LOAD keyboard command, 4-126
Load map, 4-119, 11-16
  description of sample (table),
    11-17, 11-37
  sample (figure), 11-17
Load map for program with
    extended memory overlays
    (figure), 11-36
Load map listing,
  producing a wide, 4-125, 11-49
Load module, 2-2, 11-2, 11-14
  filling unused locations in,
    11-50
  how LINK structures, 11-3
  specifying start address, 11-47
Loaded jobs,
  resouce listing, 17-5
Loaded program,
  starting, 4-171
Loading a device handler, 4-126
Loading a file into memory, 4-102
Loading BATCH, A-42
Loading ODT in memory (figure),
    21-2
Location pointer,
  current, 5-7
Locations 0-377, 11-3
Locations of bad blocks,
  finding, 4-64, 8-7
/LOG option,
  COPY, 4-37
  DELETE, 4-50
  PRINT, 4-134
  RENAME, 4-140
  TYPE, 4-174
Logging files, 7-15
Logical device names,
  assigning, 4-18
  deassigning, 4-48
Low memory, 11-2
Low memory overlay structure,
  specifying, 11-44
Low memory overlays, 11-18
  physical address space for
    program with (figure), 11-29
Lowest address used by
    relocatable code,
  specifying, 11-40

M

M command arguments (table), 5-32
/M option,
  DIR, 9-7
  LIBR, 12-14
  LINK, 11-44
  QUEMAN, 20-5
  RESORC, 17-6
$MACRO command,
  BATCH, A-26
Macro command (M),
  EDIT, 5-32
MACRO file specifications options
    (table), 10-4
MACRO keyboard command, 4-128
Macro library,
  creating, 12-14
Macro library file designation
    option, 10-8
Macro library options, 12-13
/MACRO option,
  COMPILE, 4-29
  EXECUTE, 4-86
  LIBRARY, 4-115
MACRO-11 8K version, 10-13
MACRO-11 assembler, 1-8
  calling, 10-1
  terminating, 10-3
MACRO-11 assembler options, 10-4
MACRO-11 assembler temporary work
    file, 10-3
MACRO-11 assembly language, 1-8,
    10-1
MACRO-11 error codes, 10-14
MACRO-11 error codes (table),
    10-14
MACRO-11 function control
    options, 10-7
MACRO-11 listing control options,
    10-4
MACRO-11 program assembly, 10-1
Magnetic tape, See Magtape
Magtape,
  using, 4-38, 7-8
Magtape and cassette,
  file transfers involving, 4-38,
    7-4
Magtape device conditions,
  setting, 4-155
Magtape or cassette,
  copying files to, 4-38, 7-4
Maintaining storage volumes, 8-1
Map,
  load, 11-16

/MAP option,
  EXECUTE, 4-86
  LINK, 4-122
/MATCH option,
  DIFFERENCES, 4-58
Memory,
  extended, 11-2
Memory block initialization,
  ODT, 21-16
Memory diagram showing BASIC link with overlay regions (figure), 11-26
Memory diagram showing low memory and extended memory overlays (figure), 11-35
Memory error report, 19-8
  sample (figure), 19-8
Memory error report analysis (table), 19-8
Memory image file,
  See .SAV image file
  creating a patch for, 16-5
Memory requirement,
  BL monitor, 1-3
  FB monitor, 1-3
  SJ monitor, 1-3
  XM monitor, 1-4
Memory usage,
  EDIT, 5-10
Merging library files, 12-11
Message,
  suppressing confirmation, 8-13
$MESSAGE command,
  BATCH, A-28
Minimal memory requirement, 1-3, 1-4
Modifying and examining save image files, See SIPP
Module,
  load, 2-2, 11-1, 11-13
  object, 2-2, 11-1, 11-6
Module names,
  including in a library directory, 12-7
Modules,
  adding a subroutine to, 23-6
  deleting from a library, 4-114, 12-6
  extracting from a library, 4-114, 12-6
  inserting into a library, 4-115, 7-5
  patching object, See PAT
  replacing in a library, 4-117, 12-8
Monitor,
  base-line, See BL monitor
  bootstrapping a, 4-22, 8-8
  changing, 4-22
  current,
Monitor, (Cont)
  resource listing, 17-6
  extended memory, See XM monitor
  foreground/background, See FB monitor
  single-job, See SJ monitor
  system, 1-2
Monitor commands, See keyboard commands
Monitor version identification, 3-1
Monitors,
  changing, 8-8
$MOUNT command,
  BATCH, A-29
Moving EDIT reference pointer, 5-19
Multiple copies,
  printing, 4-133, 20-5
Multiple definition global symbols, 11-7
Multiple definition libraries,
  creating, 12-10
  using, 11-14
Mutually exclusive options,
  keyboard command, 4-2

N

/N option,
  DIR, 9-7
  DUP, 8-14
  LIBR, 12-7
  PIP, 7-12
  QUEMAN, 20-6
/NAME option,
  FRUN, 4-101
  PRINT, 4-135
  SRUN, 4-169
Named p-sect, 11-6
Names,
  assigned to p-sects, 11-6
Nesting indirect command files, 4-14
/NEWFILES option,
  COPY, 4-37
  DELETE, 4-50
  DIRECTORY, 4-68
  PRINT, 4-135
  RENAME, 4-140
  TYPE, 4-175
Next command (N),
  EDIT, 5-17
/NOASCII option,
  DUMP, 4-74
/NOBITMAP option,
  LINK, 4-120
/NOCOMMENTS option,
  DIFFERENCES, 4-58

/NOEXECUTE option,
LINK, 4-121
/NOFLAGPAGE option,
PRINT, 4-134
/NOLINENUMBERS option,
COMPILE, 4-27
DIBOL, 4-53
EXECUTE, 4-85
FORTRAN, 4-94
/NOLOG option,
COPY, 4-37
PRINT, 4-135
RENAME, 4-140
TYPE, 4-174
/NOOBJECT option,
COMPILE, 4-29
DIBOL, 4-54
FORTRAN, 4-96
LIBRARY, 4-116
MACRO, 4-131
/NOOPTIMIZE option,
COMPILE, 4-29
EXECUTE, 4-87
FORTRAN, 4-96
/NOPROTECTION option,
RENAME, 4-141
/NOQUERY option,
COPY, 4-40
DELETE, 4-51
FORMAT, 4-91
INITIALIZE, 4-108
SQUEEZE, 4-168
/NOREPLACE option,
COPY, 4-40
RENAME, 4-141
/NORUN option,
EXECUTE, 4-88
/NOSHOW option,
COMPILE, 4-30
EXECUTE, 4-88
/NOSPACES option,
DIFFERENCES, 4-59
/NOSWAP option,
COMPILE, 4-30
EXECUTE, 4-88
FORTRAN, 4-97
/NOTRIM option,
DIFFERENCES, 4-61
/NOVECTORS option,
COMPILE, 4-31
EXECUTE, 4-88
FORTRAN, 4-97
/NOWARNINGS option,
COMPILE, 4-31
DIBOL, 4-55
EXECUTE, 4-89
FORTRAN, 4-98

O

/O option,
DIR, 9-7
DUP, 8-8
LINK, 11-44
PIP, 7-12
RESORC, 17-7
Object module patch program,
See PAT
Object modules, 2-2, 11-2, 11-6
examples of updating, 23-5
input to LINK, 11-11
linking, 4-119, 11-1
patching, See PAT
/OBJECT option,
COMPILE, 4-29
DIBOL, 4-54
EXECUTE, 4-87
FORTRAN, 4-95
LIBRARY, 4-116
MACRO, 4-131
Obtaining a cross-reference
table, 4-26, 4-52, 4-84,
4-128, 10-9
Obtaining system services, 1-9
/OCTAL option,
DIRECTORY, 4-68
Octal values,
entering with SIPP, 22-7
ODT, 1-7, 21-1
accessing general registers,
21-9
accessing internal registers,
21-10
back-arrow key, 21-8
calculating offsets, 21-17
calling, 21-1
changing ASCII text, 21-20
changing locations, 21-7
closing locations, 21-7
constant register, 21-16
controlling program execution,
21-12
forms of relocatable
expressions (r) (table), 21-5
functional organization, 21-22
internal registers (table),
21-10
loaded in memory (figure), 21-2
memory block initialization,
21-16
opening addressed location,
21-9
opening an individual location,
21-7
opening bytes, 21-7
opening locations, 21-7
opening previous location, 21-8

ODT, 1-7, 21-1 (Cont)
  opening subsequent locations, 21-8
  Radix-50 mode, 21-10
  relative branch offset, 21-9
  relocation, 21-4
  relocation register commands, 21-18
  return to previous sequence, 21-9
  sample load maps, 21-2
  searching for words, 21-15
  searching through program, 21-14
  single-instruction mode, 21-14
  specifying number of instructions to execute, 21-14
  suspending program execution, 21-11
  using with foreground/background jobs, 21-21
ODT breakpoints, 21-11, 21-22
ODT commands and functions, 21-6
ODT error detection, 21-26
ODT printout format, 21-6
ODT priority level, 21-19
ODT relocation calculators, 21-19
ODT searches, 21-25
ODT terminal interrupt, 21-26
On-line debugging technique, See ODT
/ONDEBUG option,
  COMPILE, 4-29
  DIBOL, 4-54
  EXECUTE, 4-87
  FORTRAN, 4-96
/ONLY option,
  DUMP, 4-74
Opening addressed location, ODT, 21-9
Opening an individual location, ODT, 21-7
Opening and closing files, 5-12
Opening bytes, ODT, 21-7
Opening locations, ODT, 21-7
Opening previous location, ODT, 21-8
Opening subsequent locations, ODT, 21-8
Operating on system files, 4-41, 4-51, 4-142, 7-13
Operating procedures, BATCH, A-42
Operator directives to BATCH run-time handler (table), A-48
Operators, SLP command file (table), 24-4
Optimization codes (table), 4-96
/OPTIMIZE option,
  COMPILE, 4-29
  EXECUTE, 4-87
  FORTRAN, 4-96
Option functions, combining library, 12-12
Options,
  assembly pass, 10-13
  BATCH command field (table), A-3
  BATCH specification field, A-7
  BINCOM, 16-2
  BINCOM (table), 16-2
  DIR, 9-2
  DIR (table), 9-2
  DUMP, 13-1
  DUMP (table), 13-1
  DUP, 8-1
  DUP (table), 8-2
  FILEX, 14-2
  FORMAT, 18-2
  FORMAT (table), 18-2
  keyboard command, 4-1
  LIBR Macro (table), 12-13
  LIBR object, 12-3
  LIBR object (table), 12-3
  linker (table), 11-9
  Macro libraries, 12-13
  MACRO-11 assembler, 10-4
  MACRO-11 listing control, 10-4
  PATCH, 25-2
  PATCH (table), 25-2
  PIP, 7-3
  QUEMAN, 20-3
  QUEMAN (table), 20-3
  RESORC, 17-1
  RESORC (table), 17-2
  SIPP, 22-2
  SIPP (table), 22-2
  SLP, 24-2
  SLP (table), 24-2
  specification field (table), A-7
  SRCCOM, 15-2
  SRCCOM (table), 15-2
  summary keyboard commands and utility program options, B-1
  SYSGEN, See Special features
/ORDER option,
  DIRECTORY, 4-68
Output files, closing, 4-24
Output format,
  BINCOM, 16-3
  SRCCOM, 15-3
Output listing, sample SRCCOM, 15-4

/OUTPUT option,
  DIFFERENCES, 4-58
  DIRECTORY, 4-70
  DUMP, 4-74
  EDIT, 4-81
  SQUEEZE, 4-167
Overlaid program segment limits (table), 22-14, 22-15
Overlay handler,
  extended memory, 11-38
  extended memory (figure), 11-38
  run-time (figure), 11-20
Overlay regions,
  memory disgram showing BASIC link with (figure), 11-26
Overlay scheme (figure), 11-20
Overlay segments, 11-2
  extending with SIPP, 22-12
Overlay structure,
  creating, 11-18
  specifying in low memory, 11-44
Overlay structure for a FORTRAN program,
  sample (figure), 11-19
Overlaying lines in a module,
  PAT, 23-5
Overlays,
  combining low memory and extended memory, 11-34
  extended memory, 11-27
  guidelines for specifying, 11-23
  load map for program with extended memory (figure), 11-36
  low memory, 11-18
  memory diagram showing low memory and extended memory (figure), 11-35
  physical address space for program with low memory (figure), 11-29
Overview,
  RT-11, I-1
OVR,
  p-sect attribute, 11-4
/OWNER option,
  COPY, 4-38
  DIRECTORY, 4-70

P

/P option,
  DIR, 9-8
  FORMAT, 18-3
  LIBR, 12-8
  LINK, 11-46
  PIP, 7-13
  QUEMAN, 20-6
  SRCCOM, 15-7
P-sect attributes, 11-4
P-sect attributes (table), 11-4
P-sect in root,
  extending, 11-41
  specifying start address, 11-49
P-sect names, 11-6
  including in library directory, 12-8
P-sect order, 11-6
P-sect ordering for FORTRAN programs, 11-6
P-sects, 11-2, 11-3, 11-5, 11-6
/PACKED option,
  COPY, 4-38
Pages, 5-1
Paging output, 3-7
Partitions,
  extended memory that contain sharing segments (figure), 11-33
/PASS:1 option,
  COMPILE, 4-30
  EXECUTE, 4-87
  MACRO, 4-131
PAT, 1-8, 23-1
  adding a subroutine to a module, 23-6
  calling, 23-1
  effecting updates, 23-3
  overlaying lines in a module, 23-5
  processing steps required to update a module using (figure), 23-2
  updating a module using (figure), 23-2
  validating contents of a file, 23-8
PAT correction file, 23-4
PAT input file, 23-4
PATCH, 1-8, 25-1
  calling, 25-1
  exiting, 25-4
  example patching a nonoverlaid file, 25-8
  including old contents in the checksum, 25-7
  patching a new file, 25-4
  setting relocation registers, 25-8
  setting the bottom address, 25-7
  setting values in the overlay handler tables, 25-7
  translating and indirectly modifying locations in a file, 25-5
Patch,
  creating for a memory image file, 16-5
  creating for a source file, 15-7

PATCH checksum, 25-2
Patch command files,
  creating, 4-56
PATCH commands, 25-2
PATCH commands (table), 25-3
PATCH control characters (table),
    25-5
PATCH examining and changing
    locations in a file, 25-4
PATCH example patching an
    overlaid file, 25-10
PATCH examples, 25-8
PATCH options, 25-1
PATCH options (table), 25-2
Patch program, See PATCH
  object module, See PAT
  save image, See SIPP
  source language, See SLP
Patching a new file,
  PATCH, 25-4
Patching object modules, See PAT
Patching pre-V04 save image
    files, See PATCH
Patching save image files,
    See SIPP
Patching source files, See SLP
/PATTERN option,
  FORMAT, 4-91
/PAUSE option,
  FRUN, 4-100
  SRUN, 4-169
Peripheral interchange program,
    See PIP
Permanent device names (table),
    3-3
Physical address space, 11-28
Physical address space for
    program with low memory
    overlays (figure), 11-29
Physical and virtual address
    space (figure), 11-30, 11-32
Physical device names (table),
    3-3
PIP, 1-6, 7-1
  calling, 7-1
  using wildcards in, 7-1
PIP options, 7-3
Pointer,
  current location, 5-7
  moving EDIT reference, 5-19
Position command (P),
  EDIT, 5-23
/POSITION option,
  COPY, 4-38
  DELETE, 4-50
  DIRECTORY, 4-70
/PREDELETE option,
  COPY, 4-40
Preventing file replace during
    copy, 4-40, 7-12
$PRINT command,
  BATCH, A-31
PRINT keyboard command, 4-133
Printer,
  line, See Line printer
/PRINTER option,
  DIFFERENCES, 4-58
  DIRECTORY, 4-70
  DUMP, 4-74
  HELP, 4-105
Printing,
  deleting input files after,
    4-133, 20-5
  resuming or restarting, 20-7
  suppressing banner pages,
    4-134, 20-6
  suspending a job, 20-7
Printing files, 4-133
Printing job banner pages, 4-134,
    20-5
Printing multiple copies, 4-133,
    20-5
Printout format,
  ODT, 21-6
Priority level,
  ODT, 21-19
Privileged jobs, 1-5, 11-31
Processing library modules, 11-12
Processing steps required to
    update a module using PAT
    (figure), 23-2
Producing a wide load map
    listing, 4-125, 11-49
Producing an LDA file, 4-122,
    11-44
Program development cycle, 2-1
Program development cycle
    (figure), 2-3
Program execution,
  controlling while debugging,
    21-12
  suspending while debugging,
    21-11
  terminating, 3-7
Program section, See P-sect
Program section order, 11-6
Program virtual address space
    (figure), 11-28
Programmed request,
  .LIMIT, 4-125, 11-49
  .SETTOP, 4-125, 11-49
Programmed requests, 1-11, II-1
Programs,
  utility, 1-5, IV-1
Prompt,
  keyboard command, 4-6
  LINK, 11-50
/PROMPT option,
  LIBRARY, 4-116
  LINK, 4-123

/PROMPT option, (Cont)
  PRINT, 4-135
Prompting characters (table), 6-3
Prompting sequence for LINK
    options (table), 4-120
Prompting sequence of LIBRARY
    options (table), 4-117
Prompts,
  keyboard command, 4-1
  keyboard monitor, 4-1
Protecting files from deletion,
    4-141, 7-15
/PROTECTION option,
  RENAME, 4-141
.PSECT MACRO-11 directive, 11-4,
    11-5
Punched cards,
  creating BATCH programs on, A-41

Q

/Q option,
  DIR, 9-8
  LINK, 11-46
  PIP, 7-15
Qualifiers, See Options
QUEMAN,
  calling, 20-2
QUEMAN options, 20-3
QUEMAN options (table), 20-3
/QUERY option,
  COPY, 4-40
  DELETE, 4-51
  FORMAT, 4-91
  INITIALIZE, 4-108
Query option,
  PIP, 7-15
/QUERY option,
  PRINT, 4-135
  RENAME, 4-141
  SQUEEZE, 4-167
  TYPE, 4-175
QUEUE, 4-133,
    See Also Queue Package
  calling, 20-1
  halting, 20-4
  listing contents of, 20-5
  removing a job from, 20-5
  resuming or restarting, 20-7
  suspending, 20-7
QUEUE option,
  SHOW, 4-165
Queue Package, 1-7, 20-1
  calling, 20-1
Queue Package work file, 20-1
Queue work file,
  deleting, 20-6
QUEUE.REL, 20-2
QUEUE.SYS, 20-2
Queuing files for printing,
    4-133, 20-1
QUFILE.TMP, 20-1
  deleting, 20-6
/QUIET option,
  DIFFERENCES, 4-59

R

R keyboard command, 4-137
/R option,
  DIR, 9-9
  DUP, 8-15
  LIBR, 12-8
  LINK, 11-47
  PIP, 7-14
  QUEMAN, 20-7
/RAD50 option,
  DUMP, 4-74
Radix-50 mode,
  ODT, 21-10
Radix-50 terminators (table),
    21-11
Radix-50 values,
  displaying and entering with
    SIPP, 22-8
Random-access devices, 3-5
Read command (R),
  EDIT, 5-15
/RECORD option,
  COMPILE, 4-30
  EXECUTE, 4-88
  FORTRAN, 4-96
Recovering deleted files, 4-44,
    8-4
REENTER keyboard command, 4-138
Registers,
  accessing general with ODT,
    21-9
  ODT internal (table), 21-10
REL,
  p-sect attribute, 11-4
REL file, 3-2
REL file format, 3-2
Relative branch offset,
  ODT, 21-9
Reloc-code, 11-4
Relocatable code,
  specifying highest address,
    11-42
  specifying lowest address used
    by, 11-40
Relocatable expressions (r),
  forms of ODT (table), 21-5
Relocatable file format, 3-2
Relocation,
  ODT, 21-4
Relocation base,
  setting, 4-20

Relocation calculators,
  ODT, 21-19
Relocation register commands,
  ODT, 21-18
REMOVE keyboard command, 4-139
/REMOVE option,
  LIBRARY, 4-116
Removing a device from the system
    tables, 4-139
Removing a job from the queue,
    20-5
Removing keyboard commands, 4-15
RENAME keyboard command, 4-140
Renaming files, 4-140, 7-14
Renaming system files, 4-142
/REPLACE option,
  COPY, 4-40
  INITIALIZE, 4-108
  LIBRARY, 4-117
  RENAME, 4-141
Replacing bad blocks, 4-108, 8-15
Replacing modules in a library,
    4-117, 12-8
Report analysis,
  error logging, 19-6
Requesting a wide listing (LIBR),
    12-9
Requests,
  programmed,
    See Programmed requests
  programmed , II-1
RESET keyboard command, 4-143
Resident monitor, See RMON
Resolving global references, 11-7
RESORC, 1-7, 17-1
  calling, 17-1
RESORC options, 17-1
RESORC options (table), 17-2
Resource listing,
  current monitor, 4-161, 17-6
  device assignments, 4-163, 17-5
  device handler status, 4-161,
    17-3
  hardware configuration, 4-161,
    17-4
  loaded jobs, 4-163, 17-5
  software configuration, 4-161,
    17-3
  special features, 4-161, 17-7
  summary, 4-161, 17-9
  terminal status, 4-165, 17-8
Resource program, See RESORC
Resources,
  showing system, 4-160, 17-1
Restarting or resuming QUEUE,
    20-7
/RESTORE option,
  INITIALIZE, 4-109
Restoring an initialized volume,
    4-109, 8-17
Restrictions,
  specifying an overlay
    structure, 11-23
RESUME keyboard command, 4-144
Resuming a suspended job, 4-144
Resuming or restarting QUEUE,
    20-7
Resuming terminal output, 3-8
Return to previous sequence,
  ODT, 21-9
/REVERSE option,
  DIRECTORY, 4-70
RO,
  p-sect attribute, 11-4
Root,
  extending p-sect in, 11-41
Root segment, 11-2
/ROUND option,
  LINK, 4-124
Rounding up size of root p-sect,
    11-48
RT-11,
  communicating with,
  BATCH, A-35
RT-11 and DOS/BATCH,
  transferring files between,
    14-3
RT-11 and interchange diskette,
  transferring files between,
    14-5
RT-11 mode,
  BATCH, A-34
RT-11 mode BATCH programs,
  creating, A-35
RT-11 mode examples,
  BATCH, A-40
RT-11 overview, I-1
RT-11 software documentation, 1-9
$RT11 command,
  BATCH, A-31
RUBOUT or DELETE, 5-3
RUBOUT or DELETE key, 3-8
Rules and conventions,
  BATCH, A-10
$RUN command,
  BATCH, A-31
RUN keyboard command, 4-145
/RUN option,
  EXECUTE, 4-88
  LINK, 4-124
Run-time overlay handler
    (figure), 11-20
Running a virtual .SAV file, 4-99
Running a virtual foreground job,
    4-99
Running BATCH, A-44
Running SIPP from a BATCH stream,
    22-17
Running SIPP from an indirect
    command file, 22-16

RW,
  p-sect attribute, 11-4

S

/S option,
  DIR, 9-9
  DUP, 8-9
  LINK, 11-47
  PIP, 7-13
  QUEMAN, 20-7
Sample assembly listing (figure), 10-6
Sample BATCH stream, A-32
Sample cache memory error report (figure), 19-8
Sample command syntax illustration (figure), 4-3
Sample load map (figure), 11-17
Sample load map description (table), 11-17, 11-37
Sample memory parity error report (figure), 19-8
Sample overlay structure for a FORTRAN program (figure), 11-19
Sample report file environment and error count report (figure), 19-10
Sample SRCCOM output listing, 15-4
Sample storage device error report (figure), 19-7
Sample storage device error report analysis (table), 19-7
Sample subroutine calls and return paths (figure), 11-23
Sample summary error report for device statistics (figure), 19-9
Sample summary error report for memory statistics (figure), 19-10
Sample text,
  SRCCOM, 15-3
SAV file format, 3-2
Save command (S),
  EDIT, 5-30
Save image files,
  patching, See SIPP
  patching pre-V04, See PATCH
Save image patch program, See SIPP
SAVE keyboard command, 4-147
Scanning for bad blocks, 4-107, 8-6
Scheme,
  overlay (figure), 11-20
Scope-code, 11-4, 11-5
Scratch region,
  creating in extended memory, 4-125, 11-49
Screen,
  display editor format for 12-inch (figure), 5-35
Screen values,
  display (table), 4-109
Search commands,
  EDIT, 5-21
Searches,
  library (figure), 11-13
  ODT, 21-25
Searching for words,
  ODT, 21-15
Searching libraries, 4-119
Searching through files,
  SIPP, 22-9
Searching while debugging, 21-14
Section attributes (table), 11-6
Sections,
  cross-reference (table), 4-129
Segment limits,
  overlaid program (table), 22-14, 22-15
Segments,
  changing number of directory, 8-14
/SEGMENTS option,
  INITIALIZE, 4-110
Selecting FORMAT verification bit patterns, 4-91, 18-3
$SEQUENCE command,
  BATCH, A-32
Sequential-access devices, 3-5
Serial line printer,
  See Also Line printer,
  See Line printer
SET device conditions and modifications (table), 4-150
SET keyboard command, 4-149
SET USR SWAP or NOSWAP, 4-158
/SETDATE option,
  COPY, 4-41
  RENAME, 4-141
Setting default number of banner pages, 20-6
Setting FORMAT verification bit patterns, 4-91, 18-3
Setting Queue Package defaults, 20-6
Setting relocation base, 4-20
Setting system configuration parameters, 4-149
Setting system date, 4-47
Setting system time, 4-173
.SETTOP programmed request, 4-125
SHOW keyboard command, 4-160
/SHOW option,
  COMPILE, 4-30

/SHOW option (Cont)
EXECUTE, 4-88
FORTRAN, 4-97
MACRO, 4-132
Simple keyboard commands, 1-9
/SINCE option,
DIRECTORY, 4-70
Single volume system,
copying files with, 7-16
Single-density mode,
formatting diskettes in, 4-91, 18-5
Single-instruction mode,
ODT, 21-14
Single-instruction mode commands,
ODT (table), 21-14
Single-job monitor,
See SJ monitor
/SINGLEDENSITY option,
FORMAT, 4-91
SIPP, 1-8, 22-1
advancing in bytes, 22-6
backing up through files, 22-6
backing up to a previous prompt, 22-11
calling, 22-1
completing code modifications, 22-12
displaying and entering ASCII values, 22-7
displaying and entering Radix-50 values, 22-8
entering octal values, 22-7
extending files and overlay segments, 22-12
running from a BATCH stream, 22-17
running from an indirect command file, 22-16
searching through files, 22-9
verifying input, 22-10
SIPP checksum, 22-15
SIPP command file,
creating, 16-5
SIPP commands, 22-4
SIPP commands (table), 22-4
SIPP dialogue, 22-3
/SIPP option,
DIFFERENCES, 4-59
SIPP options, 22-2
SIPP options (table), 22-2
Sizes,
default directory (table), 4-110
SJ monitor,
memory requirement, 1-3
/SLOWLY option,
COPY, 4-41
LINK, 4-124
SLP, 1-8, 24-1
calling, 24-1
SLP command file,
adding lines to a file, 24-6
creating, 15-7, 24-4
creating a numbered listing, 24-6
deleting lines in a file, 24-8
replacing lines in a file, 24-9
SLP command file operators (table), 24-4
SLP command file update line format, 24-4
SLP example, 24-3
/SLP option,
DIFFERENCES, 4-59
SLP options, 24-2
SLP options (table), 24-2
Software configuration,
resource listing, 4-161, 17-3
Software documentation,
RT-11, 1-9
Sort categories,
directory listing (table), 4-69
/SORT option,
DIRECTORY, 4-71
Source comparison program,
See SRCCOM
Source files,
creating a patch for, 15-7, 24-4
Source language patch program,
See SLP
/SPACES option,
DIFFERENCES, 4-59
Special features,
resource listing, 17-7
Special function keys, 3-6
Specification field options (table), A-7
Specifications,
file type, 4-5
Specifying absolute base address, 11-46
Specifying an audit trail, 4-57, 15-7
Specifying extended memory overlay structure, 11-49
Specifying highest address used by relocatable code, 11-42
Specifying lowest address used by relocatable code, 11-40
Specifying number of instructions to execute,
ODT, 21-14
Specifying start address of load module, 11-47
Specifying the stack address in link, 11-44

Sprecial features,
  resource listing, 4-161
SQUEEZE keyboard command, 4-167
Squeezing a device, 4-167, 8-9
SRCCOM, 1-7, 15-1
  calling, 15-1
SRCCOM options, 15-2
SRCCOM options (table), 15-2
SRCCOM output format, 15-3
SRCCOM sample output listing,
    15-4
SRCCOM sample text, 15-3
SRUN keyboard command, 4-169
Stack address,
  specifying in link, 11-44
/STACK option,
  LINK, 4-124
Stack pointer, 11-3
Standard file types (table), 3-4
Start address of module,
  specifying, 11-47
Start address of p-sect in root,
  specifying, 11-49
START keyboard command, 4-171
/START option,
  COPY, 4-41
  CREATE, 4-45
  DIFFERENCES, 4-59
  DIRECTORY, 4-71
  DUMP, 4-74
STARTF.COM, 4-15
Starting a loaded program, 4-171
Starting the system, 3-1
STARTS.COM, 4-15
Startup indirect command files,
    3-2, 4-15
STARTX.COM, 4-15
/STATISTICS option,
  COMPILE, 4-30
  EXECUTE, 4-88
  FORTRAN, 4-97
Storage device error report,
    4-164, 19-6
Storing volume identification,
    8-15
Structures,
  device (table), 3-6
Subroutine,
  adding to a module, 23-6
Summary,
  keyboard commands and utility
    program options, B-1
  utility program options, B-13
Summary error report, 4-164, 19-9
Summary error report for device
    statistics,
  sample (figure), 19-9
/SUMMARY option,
  DIRECTORY, 4-71
Summary resource listing, 17-9
Supported FILEX devices (table),
    14-1
Suppressing confirmation message,
    4-40, 4-51, 4-108, 4-168,
    8-13
Suppressing printing of banner
    pages, 20-6
Suppressing terminal output, 3-7
SUSPEND keyboard command, 4-172
Suspended job,
  resuming a, 4-144
Suspending execution of a job,
    4-172
Suspending program execution
    while debugging, 21-11
Suspending QUEUE, 20-7
Suspending terminal output, 3-8
/SWAP option,
  COMPILE, 4-30
  EXECUTE, 4-88
  FORTRAN, 4-97
Switches, See Options
Symbol table definition file,
    4-119
/SYMBOLTABLE option,
  LINK, 4-124
Syntax,
  CSI file specification, 6-1
  EDIT command string, 5-6
  keyboard command, 4-1
Syntax illustration,
  sample command (figure), 4-3
SYSGEN options,
    See Special features
SYSLIB, 1-11
SYSLIB-FORTRAN callable
    subprograms, 1-11
SYSMAC, See System macro library
System communication, II-1
System communication area, 11-3
System conventions, 3-1
System files,
  deleting, 4-51
  operating on, 4-142, 7-13
  renaming, 4-142
System hardware components, 1-1
System jobs, 1-7
  assigning a terminal to, 4-170
  directing control to, 3-8
  executing, 4-169
  unloading, 4-176
System macro library, 1-11,
    See SYSMAC
System monitors, 1-2
/SYSTEM option,
  COPY, 4-41
  DELETE, 4-51
  RENAME, 4-142
System program/keyboard command
    equivalents (table), B-13

System resources,
  complete listing, 4-161, 17-2
  showing, 4-160, 17-1
System services,
  obtaining, 1-9
System software components, 1-2
System subroutine library,
    See SYSLIB
System tables,
  removing a device from, 4-139
System utility programs,
    See Utility program
System volume,
  creating a bootable, 4-34, 8-11

T

/T option,
  DUP, 8-10
  GT, 4-109
  LINK, 11-47
  PIP, 7-13
  RESORC, 17-8
TAB key, 5-3
Table,
  cross-reference (figure), 10-12
TECO, 4-151
/TECO option,
  EDIT, 4-81
Temporary files,
  BATCH, A-9
Temporary work file,
  MACRO-11 assembler, 10-3
Terminal,
  assigning to a foreground job,
    4-100
  assigning to a system job,
    4-170
Terminal device conditions,
  setting, 4-156
Terminal interrupt,
  ODT, 21-26
/TERMINAL option,
  DIFFERENCES, 4-59
  DIRECTORY, 4-71
  DUMP, 4-74
  FRUN, 4-100
  HELP, 4-105
  SRUN, 4-170
Terminal output,
  resuming, 3-8
  suppressing, 3-7
  suspending, 3-8
Terminal status,
  resource listing, 4-165, 17-8
TERMINALS option,
  SHOW, 4-165
Terminating input, 3-8
Terminating program execution,
    3-7
Terminating the MACRO-11
    assembler, 10-3
Terminators,
  ASCII (table), 21-20
  Radix-50 (table), 21-11
Text editor,
  EDIT, 1-5, 4-79, 5-1
  K52, 1-5, 4-79, 4-151
  KED, 1-5, 4-79, 4-151
  TECO, 4-151
Text files,
  editing, 4-79, 5-1
  SRCCOM sample, 15-3
Text listing commands,
  EDIT, 5-24
Text modification commands,
  EDIT, 5-25
Time,
  displaying system, 4-173
  setting system, 4-173
TIME keyboard command, 4-173
/TOP option,
  LINK, 4-124
/TOPS option,
  COPY, 4-41
  DIRECTORY, 4-71
Transfer operations,
  file, See Also Copying files
/TRANSFER option,
  LINK, 4-124
Transferring files, 4-32, 7-1,
    See Also Copying files
  between RT-11 and DOS/BATCH,
    14-3
  between RT-11 and interchange
    diskette, 4-37, 14-5
  to RT-11 from DECsystem-10,
    14-8
Translating locations in a file,
  PATCH, 25-5
/TRIM option,
  DIFFERENCES, 4-61
TYPE keyboard command, 4-174
Type-ahead feature, 3-9
Type-code, 11-4
Typing additional lines of input
    to LINK, 4-123, 11-41
Typing files at terminal, 4-174

U

/U option,
  DUP, 8-11
  LIBR, 12-9
  LINK, 11-48
  PIP, 7-13
Undefined global symbols, 11-7

Underline key,
  ODT, 21-8
Uninitializing a volume, 8-17
/UNITS option,
  COMPILE, 4-30
  EXECUTE, 4-88
  FORTRAN, 4-97
UNLOAD keyboard command, 4-176
Unloading foreground jobs, 4-176
Unloading system jobs, 4-176
Unnamed CSECT,
    See Also Blank p-sect
Unresolved global symbols, 4-119
Unsave command (U),
  EDIT, 5-31
Unused areas of a volume,
  listing, 9-5
Unused locations in load module,
  filling, 11-50
Up-arrow key,
  ODT, 21-8
/UPDATE option,
  LIBRARY, 4-117
Updates,
  how PAT effects, 23-3
Updating a library file, 12-9
Updating a module using PAT
    (figure), 23-2
Updating object modules,
  examples, 23-5
User service routine, See USR
User stack, 11-3
Uses,
  error logging, 19-1
Using cassette, 4-38, 4-50, 7-4
Using display hardware, 4-103
Using ELINIT, 19-4
Using ERROUT, 19-5
Using keyboard commands, 1-9
Using magtape, 4-38, 7-8
Using ODT with
    foreground/background jobs,
    21-21
Using the assembler, 2-1, 4-86,
    4-128, 10-1
Using the debugger, 2-2
Using the editor, 2-1
Using the librarian, 2-2, 4-113,
    12-1
Using the linker, 2-2, 4-119,
    11-1
Using wildcards, 4-7, 7-1
USR, 1-2, 4-30, 4-88, 4-97, 11-6
  setting to SWAP or NOSWAP,
    4-158
Utility commands,
  EDIT, 5-30
Utility program options,
  summary, B-13
Utility program/keyboard command
    equivalents, B-13
Utility programs, 1-5, IV-1

V

/V option,
  DIR, 9-10
  DUP, 8-12, 8-14
  FORMAT, 18-5
  LINK, 11-31, 11-49
Validating contents of a file,
  PAT, 23-8
/VECTORS option,
  COMPILE, 4-30
  EXECUTE, 4-88
  FORTRAN, 4-97
Verification bit patterns,
  selecting FORMAT, 4-91, 18-3
Verify command (V),
  EDIT, 5-25
/VERIFY option,
  DIRECTORY, 4-71
  FORMAT, 4-91
Verifying SIPP input, 22-10
Verifying volumes, 4-91
Virtual .SAV file,
  running, 4-99
Virtual address space, 11-27
  program (figure), 11-28
Virtual and physical address
    space (figure), 11-30, 11-32
Virtual foreground jobs,
  running, 4-99
Virtual jobs, 1-5, 11-31
Volume directory,
    See Directory listing
Volume formatting program,
    See FORMAT
Volume identification,
  changing, 8-15
  displaying or changing, 8-12
/VOLUMEID option,
  DIRECTORY, 4-72
  INITIALIZE, 4-110
Volumes,
  bootstrapping foreign, 4-23,
    8-9
  changing during an operation,
    4-23, 4-41, 4-51, 4-72, 4-92,
    4-110, 4-136, 4-142, 7-16,
    8-13
  copying, 4-35, 8-5
  formatting, 4-90, 18-1
  initializing, 4-107, 8-14
  maintaining, 8-1
  restoring initialized, 4-109,
    8-17
  squeezing, 4-167, 8-9

W

/W option,
  DUP, 8-13
  FORMAT, 18-5
  LIBR, 12-9
  LINK, 11-49
  PIP, 7-15
/WAIT option,
  BOOT, 4-23
  COPY, 4-41
  DELETE, 4-51
  DIRECTORY, 4-72
  FORMAT, 4-92
  INITIALIZE, 4-110
Wait option,
  PIP, 7-16
/WAIT option,
  RENAME, 4-142
  SQUEEZE, 4-168
  TYPE, 4-175
/WAIT, 4-136
/WARNINGS option,
  COMPILE, 4-31
  DIBOL, 4-55
  EXECUTE, 4-89
  FORTRAN, 4-97
Wide listing,
  requesting (LIBR), 12-9
Wide load map listing,
  producing, 4-125, 11-49
/WIDE option,
  EXECUTE, 4-89
  LINK, 4-125
Wild card construction,
  BATCH, A-6
Wildcard defaults (table), 4-8
Wildcards,
  commands supporting (table), 4-8
  using, 4-7, 7-1
Word 56 of .SAV file,
  inserting value into, 11-43
Words,
  searching for while debugging, 21-15
/WORDS option,
  DUMP, 4-74
Work file,
  queue,
  deleting, 20-6
  Queue Package, 20-1
Write command (W),
  EDIT, 5-16
Write command arguments (table), 5-17

X

/X option,
  LIBR, 12-10
  LINK, 11-49
XM monitor, 1-4
  memory requirement, 1-4
XM overlays,
  See Extended memory overlays
/XM, 4-125

Y

/Y option,
  DUP, 8-13
  LINK, 11-49
  PIP, 7-13

Z

/Z option,
  DUP, 8-14
  LINK, 11-50
  PIP, 7-15
  RESORC, 17-9
Zeroing a volume directory,
  4-107, 8-14

READER'S COMMENTS

NOTE: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. If you require a written reply and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

Did you find errors in this manual? If so, specify the error and the page number.

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
- ☐ Other (please specify)_____

Name_____ Date_____

Organization_____

Street_____

City_____ State_____ Zip Code or Country_____

Do Not Tear - Fold Here and Tape

digital

No Postage Necessary if Mailed in the United States

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO.33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

BSSG PUBLICATIONS TW/A14
DIGITAL EQUIPMENT CORPORATION
1925 ANDOVER STREET
TEWKSBURY, MASSACHUSETTS 01876

Do Not Tear - Fold Here