

# Introduction to RT-11

AA-5281C-TC

**March 1983**

This document is an introductory manual for the RT-11 operating system. Its purpose is to acquaint new users with the RT-11 commands that perform common system operations. This manual presents the background material necessary to understand the system operations. It also contains a series of command examples and demonstration exercises that complement the text.

This manual supersedes the *Introduction to RT-11*, AA-5281B-TC. This manual contains Update Notice 1, AD-5281C-T1.

**Operating System:** RT-11 Version 5.1

To order additional documents from within DIGITAL, contact the Software Distribution Center, Northboro, Massachusetts 01532.

To order additional documents from outside DIGITAL, refer to the instructions at the back of this document.

First Printing, March 1980  
Revised, March 1983  
Updated, July 1984

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

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

No responsibility is assumed for the use or reliability of software on equipment that is not supplied by DIGITAL or its affiliated companies.

© Digital Equipment Corporation 1980, 1983, 1984.  
All Rights Reserved.

Printed in U.S.A.

A postage-paid READER'S COMMENTS form is included on the last page of this document. Your comments will assist us in preparing future documentation.

The following are trademarks of Digital Equipment Corporation:

DEC  
DECmate  
DECsystem-10  
DECSYSTEM-20  
DECUS  
DECwriter  
DIBOL

**digital**™  
MASSBUS  
PDP  
P/OS  
Professional  
Rainbow  
RSTS  
RSX

RT-11  
UNIBUS  
VAX  
VMS  
VT  
Work Processor

# CONTENTS

	Page
<b>PREFACE</b>	ix
<b>CHAPTER 1 INTRODUCING THE RT-11 COMPUTER SYSTEM</b>	<b>1-1</b>
SYSTEM HARDWARE	1-1
The Computer	1-1
The Terminal	1-3
The Storage Medium	1-4
Optional Devices	1-6
SYSTEM SOFTWARE	1-7
The RT-11 Operating System	1-8
Language Processors	1-10
Application Packages	1-10
SYSTEM DOCUMENTATION	1-10
Hardware Manuals	1-10
Software Manuals	1-11
Source Listings	1-11
<b>CHAPTER 2 STARTING THE RT-11 COMPUTER SYSTEM.</b>	<b>2-1</b>
COMPUTER MEMORY	2-1
HARDWARE CONFIGURATION	2-1
Terminal	2-3
Computer	2-3
System Volume	2-3
Storage Volume	2-4
Optional Devices and Supported Languages	2-4
BOOTSTRAP PROCEDURE	2-4
<b>CHAPTER 3 INTERACTING WITH THE RT-11 COMPUTER SYSTEM</b>	<b>3-1</b>
USING THE CONSOLE TERMINAL TO EXCHANGE INFORMATION	3-1
USING MASS STORAGE VOLUMES	3-4
File Storage	3-7
File Protection	3-7
<b>CHAPTER 4 USING THE MONITOR COMMAND LANGUAGE.</b>	<b>4-1</b>
ENTERING COMMAND INFORMATION	4-1
General Command Format	4-2
Control Commands	4-3
Re-Creating the Examples	4-3

CORRECTING TYPING MISTAKES. . . . .	4-4
INITIAL MONITOR COMMAND OPERATIONS. . . . .	4-5
Using VT11 Display Hardware . . . . .	4-5
Entering the Date and Time-of-Day . . . . .	4-8
Assigning Logical Names to Devices. . . . .	4-9
Listing Volume Directories . . . . .	4-12
Initializing the Storage Volume . . . . .	4-15
<b>CHAPTER 5 CREATING AND EDITING TEXT FILES . . . . .</b>	<b>5-1</b>
THE RT-11 EDITOR . . . . .	5-1
CREATING A TEXT FILE. . . . .	5-2
EDITING A TEXT FILE . . . . .	5-4
USING UPPERCASE AND LOWERCASE CHARACTERS. . . . .	5-12
USING A GRAPHICS DISPLAY TERMINAL DURING EDITING . . . . .	5-15
Normal Use of the Graphics Display. . . . .	5-16
Immediate Mode . . . . .	5-16
CREATING THE DEMONSTRATION PROGRAMS . . . . .	5-19
<b>CHAPTER 6 COMPARING TEXT FILES . . . . .</b>	<b>6-1</b>
PERFORMING A COMPARISON . . . . .	6-1
<b>CHAPTER 7 PERFORMING FILE MAINTENANCE     OPERATIONS . . . . .</b>	<b>7-1</b>
FILE DIRECTORY OPERATIONS. . . . .	7-1
MULTIPLE FILE OPERATIONS . . . . .	7-2
FILE COPYING OPERATIONS . . . . .	7-3
FILE RENAMING OPERATIONS . . . . .	7-4
FILE DELETION OPERATIONS . . . . .	7-5
FILE PROTECTION OPERATIONS . . . . .	7-7
FILE LISTING OPERATIONS . . . . .	7-8
<b>CHAPTER 8 CHOOSING A PROGRAMMING     LANGUAGE. . . . .</b>	<b>8-1</b>
HIGH-LEVEL VS MACHINE-LEVEL LANGUAGES . . . . .	8-1
RT-11 PROGRAMMING LANGUAGES . . . . .	8-3
CHOOSING A LANGUAGE FOR THE DEMONSTRATION . . . . .	8-4
<b>CHAPTER 9 RUNNING A FORTRAN IV PROGRAM. . . . .</b>	<b>9-1</b>
DEVELOPING AN EXECUTABLE FORTRAN IV PROGRAM . . . . .	9-1
USING THE FORTRAN IV LANGUAGE PROCESSOR . . . . .	9-1

USING LIBRARY MODULES . . . . .	9-2
COMPILING THE FORTRAN IV PROGRAM . . . . .	9-3
LINKING OBJECT MODULES TOGETHER. . . . .	9-8
RUNNING THE FORTRAN IV PROGRAM . . . . .	9-11
COMBINING OPERATIONS. . . . .	9-12
ALTERNATE FUNCTIONS . . . . .	9-13
FILE MAINTENANCE . . . . .	9-14
<b>CHAPTER 10 RUNNING A BASIC-11 PROGRAM . . . . .</b>	<b>10-1</b>
DEVELOPING A BASIC-11 PROGRAM. . . . .	10-1
USING THE BASIC-11 LANGUAGE PROCESSOR . . . . .	10-1
USING THE BASIC INTERPRETER. . . . .	10-2
Immediate Mode . . . . .	10-3
Creating and Editing a BASIC-11 Program . . . . .	10-4
RUNNING A BASIC-11 PROGRAM. . . . .	10-8
FILE MAINTENANCE . . . . .	10-12
<b>CHAPTER 11 RUNNING A MACRO-11 ASSEMBLY LANGUAGE PROGRAM . . . . .</b>	<b>11-1</b>
DEVELOPING A MACRO-11 ASSEMBLY LANGUAGE PROGRAM . . . . .	11-1
USING THE MACRO-11 LANGUAGE PROCESSOR . . . . .	11-2
The Program Counter . . . . .	11-3
The Symbol Table . . . . .	11-4
Machine Language Code . . . . .	11-4
ASSEMBLING THE MACRO-11 PROGRAM . . . . .	11-6
LINKING OBJECT MODULES TOGETHER. . . . .	11-13
RUNNING THE MACRO-11 PROGRAM . . . . .	11-15
COMBINING OPERATIONS. . . . .	11-16
FILE MAINTENANCE . . . . .	11-18
<b>CHAPTER 12 LINKING OBJECT PROGRAMS . . . . .</b>	<b>12-1</b>
RESOLVING SYMBOLIC AND LIBRARY REFERENCES. . . . .	12-2
PROGRAM RELOCATION AND ADDRESS ALIGNMENT . . . . .	12-3
Absolute and Relocatable Program Sections . . . . .	12-4
The Overlay Feature . . . . .	12-6
PRODUCING A LOAD MODULE AND A LOAD MAP . . . . .	12-7
<b>CHAPTER 13 CONSTRUCTING LIBRARY FILES . . . . .</b>	<b>13-1</b>
KINDS OF LIBRARY FILES. . . . .	13-1
Macro Libraries . . . . .	13-1

Object Libraries . . . . .	13-1
CREATING AND MAINTAINING A LIBRARY FILE . . . . .	13-2
Creating Object Library Input Files . . . . .	13-2
Building the Object Library . . . . .	13-5
Updating the Object Library. . . . .	13-6
FILE MAINTENANCE . . . . .	13-7
<b>CHAPTER 14  DEBUGGING A USER PROGRAM.</b> . . . . .	<b>14-1</b>
AVOIDING PROGRAMMING ERRORS . . . . .	14-1
WHEN PROGRAMMING ERRORS OCCUR . . . . .	14-2
USING THE ON-LINE DEBUGGING TECHNIQUE . . . . .	14-3
FILE MAINTENANCE . . . . .	14-13
<b>CHAPTER 15  USING THE FOREGROUND/BACKGROUND</b> <b>MONITOR</b> . . . . .	<b>15-1</b>
THE FOREGROUND/BACKGROUND ENVIRONMENT . . . . .	15-1
Running the Foreground/Background Programs . . . . .	15-1
Creating the Background Job . . . . .	15-2
Editing the Background Job . . . . .	15-2
Running the Background Job . . . . .	15-2
USING THE FB MONITOR . . . . .	15-3
Communication in a Two-Job Environment . . . . .	15-3
Creating the Foreground Job . . . . .	15-4
Executing the Foreground and Background Jobs. . . . .	15-5
FILE MAINTENANCE . . . . .	15-8
<b>CHAPTER 16  USING INDIRECT FILES.</b> . . . . .	<b>16-1</b>
CREATING AN INDIRECT FILE . . . . .	16-1
Entering Monitor Commands . . . . .	16-1
Using the Editor to Create an Indirect File . . . . .	16-2
EXECUTING AN INDIRECT FILE . . . . .	16-4
FILE MAINTENANCE . . . . .	16-8
<b>CHAPTER 17  ADVICE TO NEW USERS.</b> . . . . .	<b>17-1</b>
USING THE HELP FILE . . . . .	17-2
<b>APPENDIX A  MANUAL BOOTSTRAPPING</b> <b>OPERATIONS.</b> . . . . .	<b>A-1</b>
BOOTSTRAPPING THE SYSTEM . . . . .	A-1
TYPING THE BOOTSTRAP ON THE TERMINAL KEYBOARD . . . . .	A-3
USING A PUSHBUTTON CONSOLE TO BOOTSTRAP . . . . .	A-4

USING A SWITCH REGISTER CONSOLE TO BOOTSTRAP. . . . .	A-5
--	-----

**APPENDIX B SELECTED SYSTEM TOPICS . . . . . B-1**

<b>STOPPING AND STARTING</b>	
THE SYSTEM . . . . .	.B-1
Stopping the System. . . . .	.B-1
Starting the System . . . . .	.B-1
THE SYSTEM STOPS UNEXPECTEDLY . . . . .	.B-2
<b>SUGGESTIONS FOR BOOTSTRAPPING</b>	
THE SYSTEM . . . . .	.B-2
BACKING UP THE SYSTEM VOLUME. . . . .	.B-3
DIRECTORY- VS NONDIRECTORY- STRUCTURED VOLUMES . . . . .	.B-4
ALTERNATE RENAME OPERATION FOR MAGTAPE USERS. . . . .	.B-4
USING THE FORTRAN/BASIC LANGUAGE VOLUME . . . . .	.B-5
SUBSTITUTING VOLUMES DURING OPERATIONS. . . . .	.B-6
USING THE LINK VOLUME . . . . .	.B-8
FORTRAN/LINK FILE MAINTENANCE . . . . .	.B-9

<b>GLOSSARY . . . . .</b>	<b>Glossary-1</b>
---------------------------	-------------------

<b>INDEX. . . . .</b>	<b>Index-1</b>
-----------------------	----------------

**FIGURES**

<b>FIGURE 1</b>	<b>Flowchart for Selective Reading . . . . .</b>	<b>xii</b>
1-1	RT-11 Computer System . . . . .	1-2
1-2	PDP-11 Computers . . . . .	1-3
1-3	Terminal Devices . . . . .	1-4
1-4	Randon-Access Storage Media and Their Devices . . . . .	1-5
1-5	Peripheral Devices . . . . .	1-6
1-6	RT-11 System Software . . . . .	1-8
1-7	RT-11 Operating System . . . . .	1-9
2-1	Bootstrap/Computer Relationship . . . . .	2-2
3-1	LA120/VT100 Terminals . . . . .	3-2
3-2	LA120/VT100 Keyboard Layouts . . . . .	3-3
3-3	Mass Storage Volumes . . . . .	3-6
4-1	VT11 Display Hardware. . . . .	4-6
5-1	Editing with RT-11 . . . . .	5-2
5-2	Text Window Format . . . . .	5-16
9-1	Evolution of a FORTRAN IV Program. . . . .	9-1
9-2	Function of a FORTRAN IV Compiler . . . . .	9-2

9-3	Dimensions of FUN(X,Y) . . . . .	9-7
9-4	The Link Operation . . . . .	9-9
9-5	The Result of GRAPH.SAV . . . . .	9-12
10-1	Functions of the BASIC-11 Language Processor . . . . .	10-2
11-1	PDP-11 Programming Card . . . . .	11-1
11-2	Evolution of a MACRO-11 Program . . . . .	11-2
11-3	Function of a MACRO-11 Assembler . . . . .	11-2
11-4	PDP-11 Computer Memory . . . . .	11-3
11-5	PDP-11 Word . . . . .	11-5
11-6	The Link Operation . . . . .	11-14
12-1	Link Functions . . . . .	12-1
12-2	Object Module Relocation . . . . .	12-4
A-1	Pushbutton Console . . . . .	A-4
A-2	Switch Register Consoles . . . . .	A-5

## TABLES

TABLE	2-1	Representative System Volumes . . . . .	2-4
	2-2	Bootstrap Prompts and Responses . . . . .	2-6
	3-1	Keyboard Characters . . . . .	3-5
	4-1	Keyboard Symbols . . . . .	4-4
	4-2	Physical Device Names . . . . .	4-9
	4-3	Special Logical Device Names . . . . .	4-10
	4-4	File Types . . . . .	4-13
	5-1	Command Arguments . . . . .	5-5
	5-2	Immediate Mode Commands . . . . .	5-17
	8-1	Language Comparisons . . . . .	8-2
	11-1	Decimal/Octal/Binary Conversion . . . . .	11-6
	A-1	Binary Conversion . . . . .	A-6

## PREFACE

The RT-11 (Real Time-11) computer system is a single-user computer/operating system that serves the programming needs of both the beginning and the advanced programmer. RT-11 supports a number of programming languages, including industry-standard FORTRAN and BASIC; and — for more advanced users — the PDP-11 assembly language, MACRO-11. RT-11 also provides a comprehensive set of operating commands for controlling system operations.

The purpose of this introductory manual is to acquaint you with a number of RT-11 operating commands that are used to perform common system operations. The manual first presents information that you need to understand a particular system operation; then it shows you how to apply the system operation in a series of operating commands and exercises that you recreate; finally, it provides a list of reference materials that contain more information about the operation. This approach makes it possible for you to learn quickly the major features of the system; at the same time, it eliminates many of the learning problems encountered by new users.

This manual describes system usage fundamentals. It is not the intent of this manual to teach you to program the PDP-11 computer. You may already be proficient in one or more of the available programming languages. Likewise, no attempt has been made in this manual to cover all the possible applications for which the RT-11 computer system is suited. You will discover many applications yourself as you continue to use the system.

This manual is designed for three categories of RT-11 users:

- **Inexperienced users:** Those having little or no previous “hands-on” computer experience (including those whose experience has been limited to batch environments)
- **Experienced users:** Those who are experienced users of a computer system other than the RT-11 computer system
- **Experienced RT-11 users:** Those who have used previous versions of the RT-11 computer system but wish a quick introduction to the newest features of the current system (Version 5)

The manual contains 17 chapters and 2 appendixes. The descriptions that follow and the chart at the end of this section will help you determine your own reading path.

## MANUAL INTENT

## MANUAL DESIGN

Chapter 1, *Introducing the RT-11 Computer System*, discusses general system concepts. It introduces the roles of hardware and software in a computer system and describes the specific hardware and software components of the RT-11 computer system. Chapter 1 is intended for users in the first two categories.

Chapter 2, *Starting the RT-11 Computer System*, shows all users how to start the system.

Chapter 3, *Interacting with the RT-11 Computer System*, demonstrates how you use the console terminal to control system operations. Again, this chapter is most helpful to users in the first two categories.

Chapters 4 through 7 describe system operations that are useful to all categories of users. Each chapter begins with a textual explanation of a particular system operation and expands into computer demonstrations showing the operation in use. Topics covered are: Using the Monitor Command Language; Creating and Editing Text Files; Comparing Text Files; and Performing File Maintenance Operations. Experienced RT-11 users may prefer to skip the textual explanations and review only the computer exercises.

Chapter 8, *Choosing a Programming Language*, helps you determine which language to use. Choose BASIC-11, FORTRAN IV, MACRO-11, or a combination of these three languages to continue the exercises in this manual (BASIC-11 and FORTRAN IV are optional products).

Chapters 9, 10, and 11 describe the process of running programs written in the FORTRAN IV, BASIC-11, and MACRO-11 languages, respectively. You should read any chapters that apply to your choice of language.

MACRO-11 and FORTRAN IV users should continue to Chapter 12, *Linking Object Programs*, and Chapter 13, *Constructing Library Files*.

Chapter 14, *Debugging a User Program*, provides some suggestions for finding and fixing errors in user programs; all users should read this chapter.

Chapter 15, *Using the Foreground/Background Monitor*, is intended for users who plan to exercise the foreground/background capability of the RT-11 system.

All users should continue to Chapter 16, *Using Indirect Files*. Indirect files allow the system to perform operations unattended.

Chapter 17 gives some advice to new users and includes a description of the RT-11 HELP file.

Two appendixes are provided for reference. Appendix A discusses manual bootstrapping procedures and Appendix B provides some additional information on selected system usage.

A glossary of technical terms appears at the end of the manual for reference purposes.

The following flowchart will help you plan your reading path through the manual. Read the chart from top to bottom; answer the questions and follow the direction of the arrows to see which chapters you should read.

#### NOTE

The demonstration portions of this manual are for use with Version 5 and later releases of RT-11. The exercises are quite lengthy, and you may prefer not to complete them in one sitting. You may pause at the end of any individual chapter. It is recommended that you stop only at the end of a chapter since you will otherwise not complete an exercise and thus may introduce errors that will affect later exercises. Instructions for pausing and beginning again are given in Appendix B.

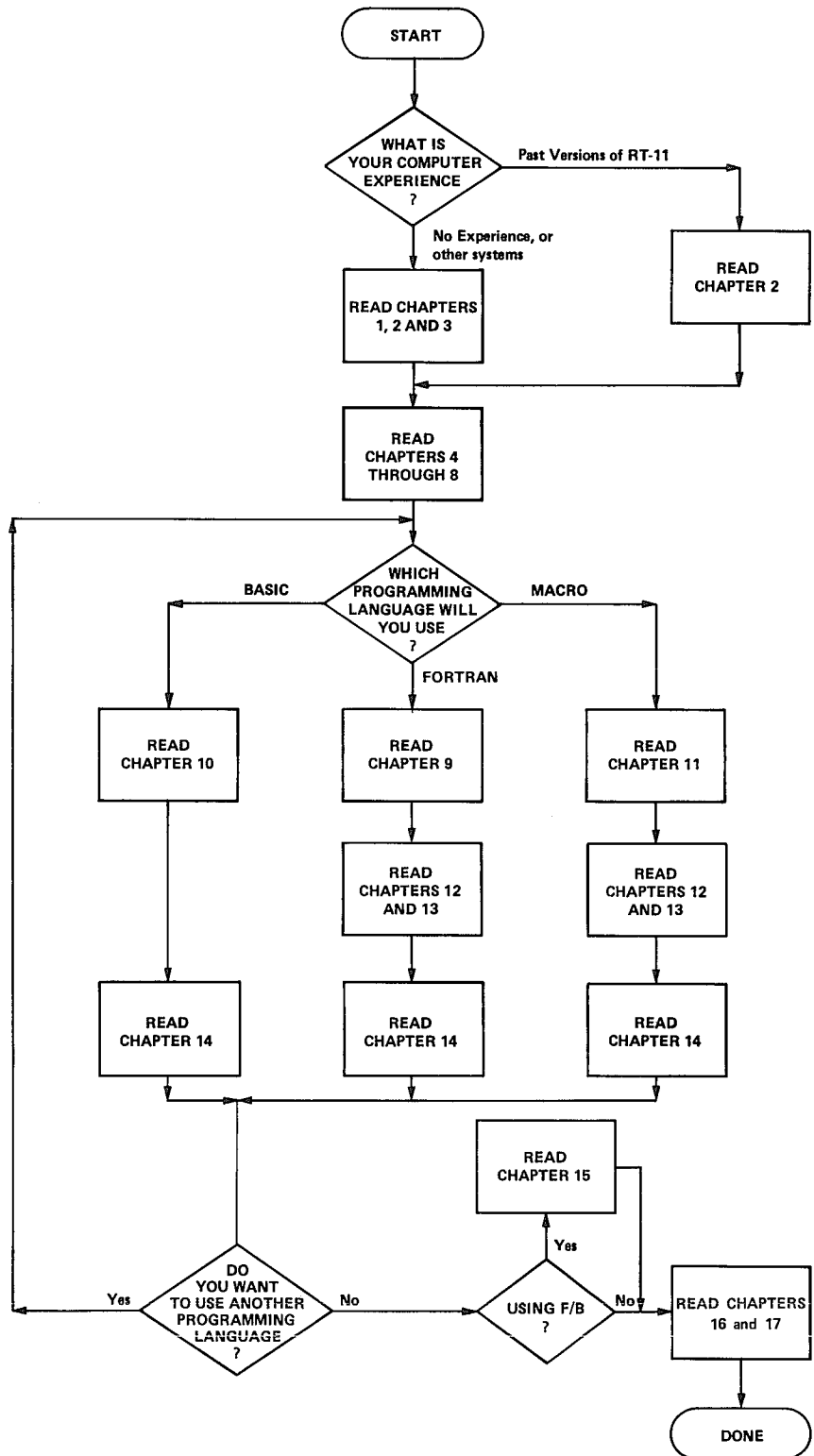


Figure 1 Flowchart for Selective Reading

# CHAPTER 1

## INTRODUCING THE RT-11 COMPUTER SYSTEM

A computer system is a collection of components that work together to process data. The purpose of a computer system is to make it as easy as possible for you to use a computer to solve problems. A functioning computer system combines hardware elements with software elements. The hardware elements are the mechanical devices in the system, the machinery and the electronics that perform physical functions. The software elements are the programs written for the system; these programs perform logical and mathematical operations and provide a means for you to control the system. Documentation includes the manuals and listings that tell you how to use the hardware and software. Collectively, these components provide a complete computer system that allows both layman and expert alike to use a computer.<sup>1</sup>

SYSTEM HARDWARE  
SYSTEM SOFTWARE  
+ SYSTEM DOCUMENTATION  
COMPUTER SYSTEM

The RT-11 computer system requires three basic hardware items: the computer, which performs all data processing; a terminal device, used like a typewriter for two-way communication between the user and the system; and a storage medium, for storing programs and data. Figure 1-1 illustrates the hardware components of a typical RT-11 computer system.

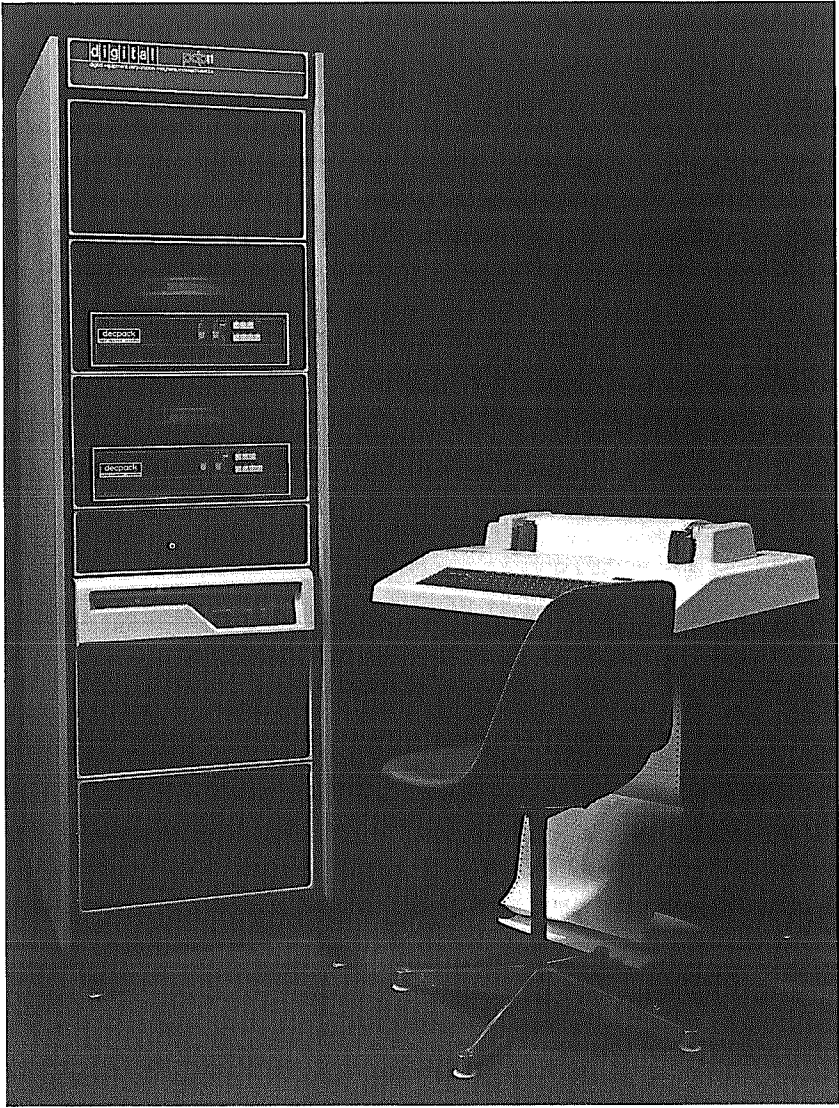
### SYSTEM HARDWARE

The computer performs all instruction decoding and data processing. The RT-11 computer system is constructed around a DIGITAL PDP-11 computer, several of which are shown in Figure 1-2. Any model of PDP-11 can be used in an RT-11 system.

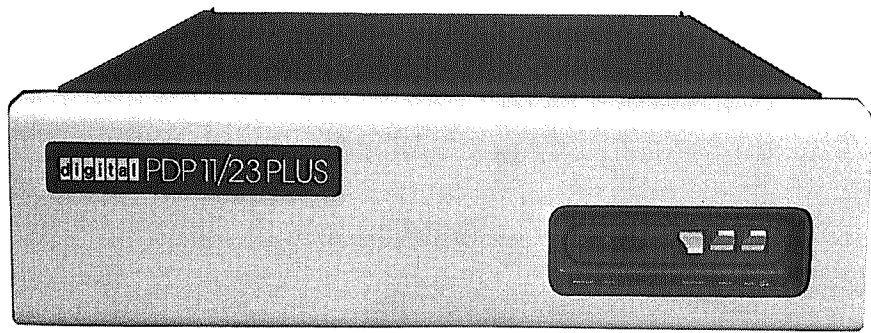
### The Computer

Notice in Figure 1-2 that the front panel, or operator's console, of each PDP-11 computer is slightly different. The switches, buttons, and lights that are on the operator's console are used for various kinds of computer operations and applications. In the RT-11 computer system they are used only to start the system. Once the system has been started, your interaction with the computer system occurs through the terminal.

<sup>1</sup>This chapter attempts to build a working vocabulary that is both meaningful to the new user and consistent with standard DIGITAL terminology. Some definitions may appear inconsistent with those you have previously learned or used.



**Figure 1-1 RT-11 Computer System**



**PDP11/23 PLUS**



**PDP11/23**



**PDP 11/44**

**Figure 1-2 PDP-11 Computers**

The terminal allows two-way communication between you (the user) and the computer system. You enter information — operating commands, for example — from the terminal keyboard, which is operated much like a typewriter keyboard. The computer, in turn, prints information and messages on the terminal's printer or screen. Figure 1-3 shows two terminal devices — the VT100 video terminal and the LA120 hardcopy terminal — that can be used in an RT-11 computer system.

## **The Terminal**



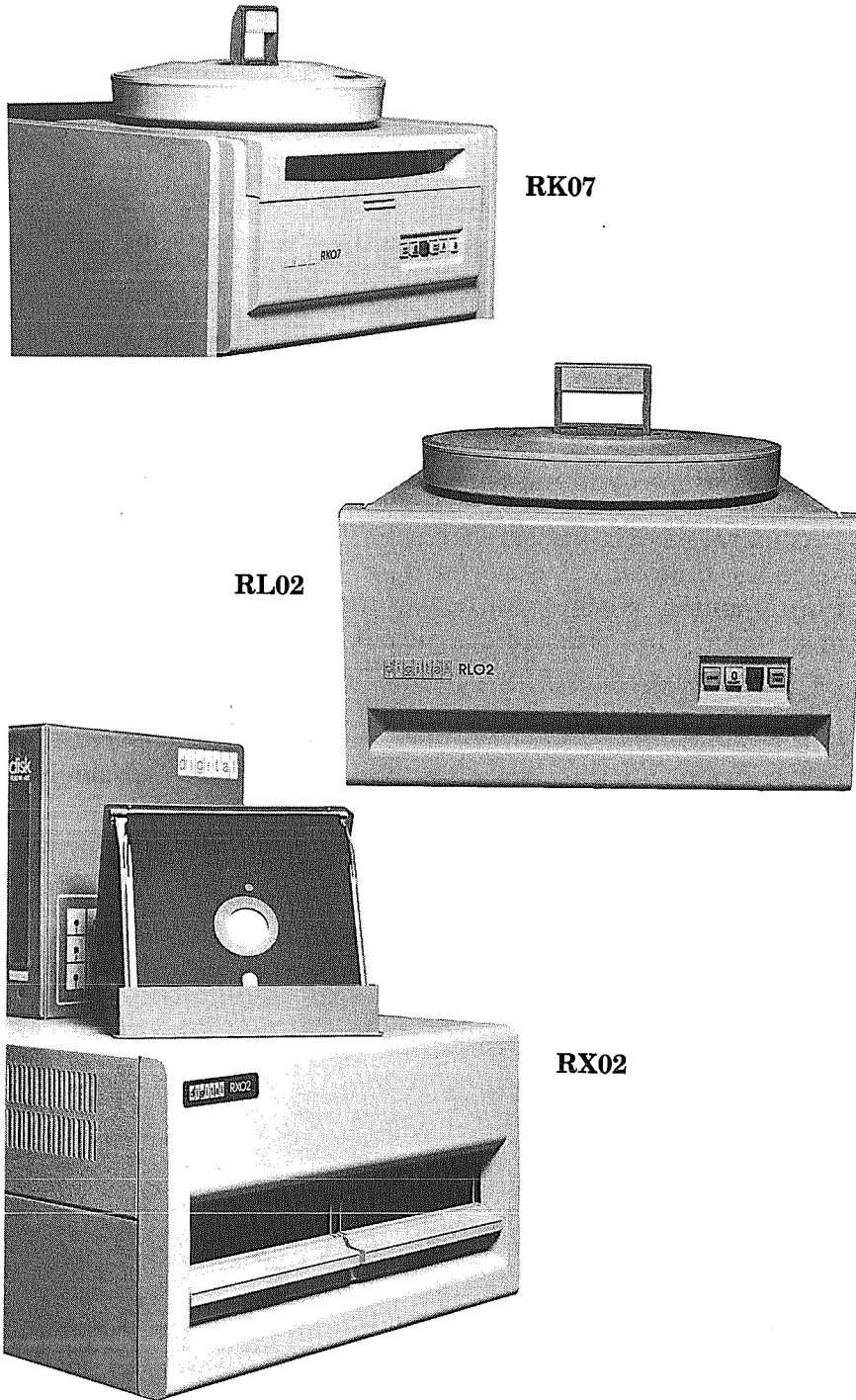
**Figure 1-3 Terminal Devices**

Generally, an RT-11 computer system has only one terminal through which all system/user interaction takes place. This is called the console terminal. If the system has more than one terminal, one of them is still designated the console terminal; others simply provide auxiliary message-printing capabilities.

### **The Storage Medium**

The third important hardware device in an RT-11 computer system is the storage medium (usually a disk). It stores programs — those that make up the computer system software and those that you create. It serves as a distribution medium; system software is often packaged and distributed on a disk by the system supplier. Finally, it stores other data, information that is eventually needed for a computer operation (called input), the results of a computer operation (called output), or

textual information such as a report. Figure 1-4 shows the random-access storage media (and their specific drive units) that can be used in an RT-11 computer system. (Random access means that access time for data is independent of the location of data. Contrast this concept with sequential access.)



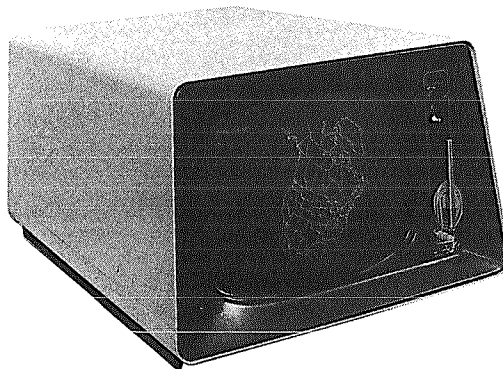
**Figure 1-4 Random-Access Storage Media and Their Devices**

These three devices — the computer, the terminal, and the storage medium — are the required hardware components of an RT-11 computer system. With the exception of the computer, all hardware devices are called peripheral devices. Peripheral devices supplement the computer by providing external resources for operations that the computer cannot handle alone. In addition to the terminal and storage medium (which are required peripheral devices), other peripheral devices can be used in an RT-11 computer system.

## **Optional Devices**

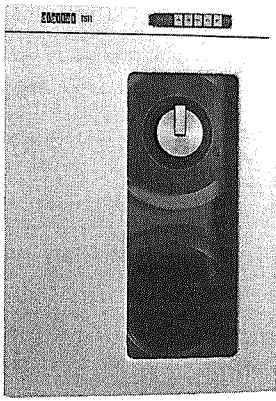
Optional peripheral devices are added to a computer system according to the specific needs of the system users. For example, computer systems that are used primarily for program development may have extra storage devices and a high-speed printing device. Computer systems used in a laboratory environment may have graphics display hardware, an oscilloscope device, and an analog-to-digital converter. Computer systems that provide (or use) information in conjunction with another kind of computer system usually have a magtape device, because magtape is an industry-standard storage device.

Peripheral devices are categorized as input/output (I/O) devices since the functions they perform provide information (input) to the computer, accept information (output) from the computer, or do both. Line printers are output devices because they perform only output operations. Terminals and storage devices are input/output devices because they perform both input and output operations. Figure 1-5 shows several of the optional peripheral devices that are often added to an RT-11 computer system.

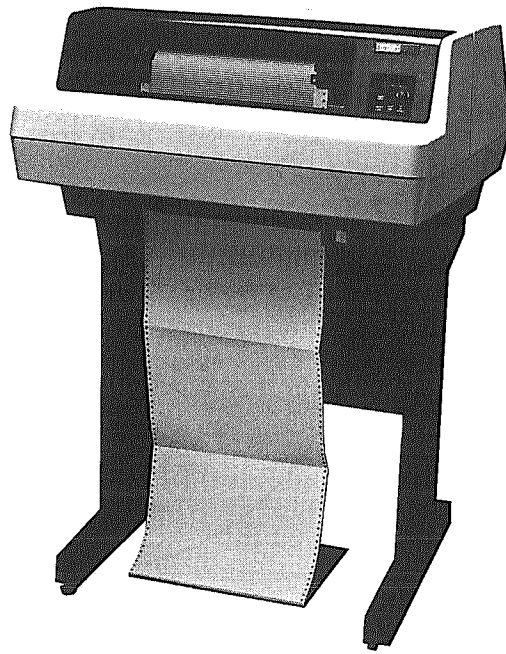


**VT11 Display**

**Figure 1-5 Peripheral Devices**



**Magtape**



**Line Printer**

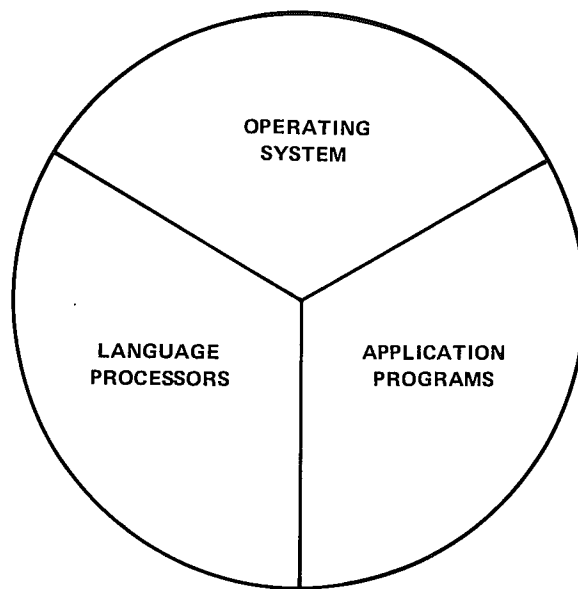
**Figure 1-5 Peripheral Devices (Cont.)**

The hardware configuration of your own RT-11 computer system includes the computer, the terminal, the storage medium, and any other peripheral devices you choose to add.

System software is an organized set of supplied programs that effectively transform the system hardware components into usable tools. These programs include operations, functions, and routines that make it easier for you to use the hardware to solve problems and produce results. For example, some system programs store and retrieve data among the various peripheral devices. Others perform difficult or lengthy mathematical calculations. Some programs allow you to create, edit, and process application programs of your own. Still others handle entire applications for you.

As illustrated in Figure 1-6, system software always includes an operating system, which is the "intelligence" of the computer system. Usually the system software includes one or several language processors; it sometimes also includes specific applications.

## **SYSTEM SOFTWARE**



**Figure 1-6 System Software**

## **The RT-11 Operating System**

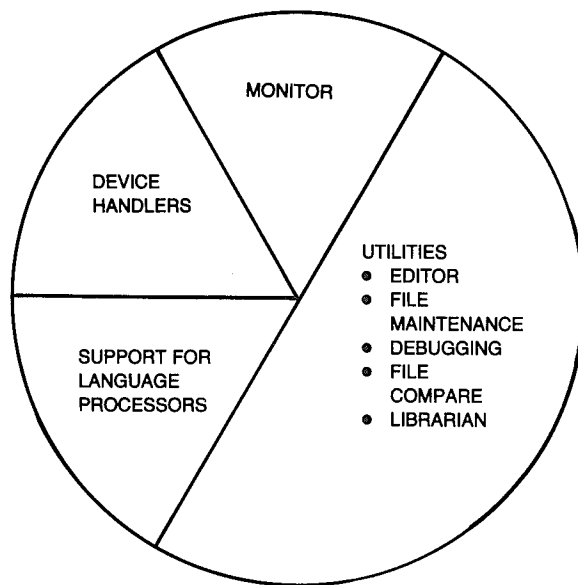
An operating system is a collection of programs that provides an environment in which you can create and run programs of your own. The operating system organizes all the hardware and software resources of the computer system into a working unit and gives you control.

The RT-11 operating system comprises four types of programs: a monitor/executive program for system control and supervision; several device handlers (programs), one for each of the supported hardware devices; a variety of utility programs for program/data creation and manipulation; and finally, the interfaces that are necessary to support several programming language processors. The operating system is illustrated in Figure 1-7.

The monitor (executive) program is the link between the system hardware, the system software, and you. Part of the monitor function is to accept, process, and execute your instructions for controlling the system. A comprehensive set of monitor operating commands allows you to direct, from the console terminal keyboard, those system operations that you want to occur.

Device handlers are routines that provide the interface to the various hardware devices that are part of the computer system. A handler exists for every peripheral device that the system supports.

Utility programs cover a wide range of resources; such programs allow you to create and edit text, maintain other pro-



**Figure 1-7 The RT-11 Operating System**

grams, and locate user-programming errors. Some utility programs in the RT-11 operating system are the following:

- An editor, which allows you to create and modify textual material; this material could be the statements that make up a computer program, a memo, or any text you wish to create
- File maintenance utility programs, which allow you to manipulate and maintain your programs and data — to transfer them between devices, to update them, and to delete them when you are done with them
- A debugging program, which helps you uncover and correct errors in your programs
- A librarian, which makes it easy for you to store and retrieve often-used programming routines
- A linking program, which converts object modules into a format suitable for loading and execution
- A source comparison program, which is used to compare two ASCII files and to output any differences to a specified output device
- A dump program, which outputs to the console or line printer all or any part of a file in octal words, octal bytes, ASCII characters, or Radix-50 characters

The RT-11 operating system also provides support for several programming languages and their respective language processors.

## **Language Processors**

A language processor is a translating program that you use to process a source program you have created. A language processor exists for every programming language supported by the system, whether it is a high-level language or a machine-level language.<sup>1</sup>

High-level languages, such as BASIC-11 and FORTRAN IV, are relatively easy languages to learn and use. Since a single language statement often performs a series of intricate computer operations, high-level languages let you direct your attention to solving the problem at hand. They do not require that you understand how the computer interprets the problem. In addition to FORTRAN IV and BASIC-11, the RT-11 operating system supports the high-level language DIBOL, DIGITAL's interactive commercial language.

Machine-level or assembly languages are available for users who prefer to work at the instruction level of the computer. At this level, you have control over such factors as program size and speed of execution. Machine-level languages do require that you be familiar with the computer and the hardware devices of the system. RT-11 provides the MACRO-11 assembly language processor for those who would rather work at this more intricate level.

## **Applications Packages**

The RT-11 operating system supports several applications packages. These include a laboratory applications package for the standard functions found in most laboratory environments. A scientific subroutine package (for FORTRAN IV users) provides a large selection of mathematical and statistical routines commonly required in scientific programming. And a graphics support package for BASIC-11 and FORTRAN IV users provides display features such as multiple intensity and blinking vectors (lines), alphanumerics, and points. Because of the specialized nature of these applications packages, they are not described further in this manual.

## **SYSTEM DOCUMENTATION**

The third component of a computer system is documentation, which includes manuals that tell you how to use the software and hardware of the computer system. Documentation also includes any source listings of programs that make up the operating system.

## **Hardware Manuals**

Hardware manuals describe the devices in the computer system. RT-11 hardware documentation includes a Processor

---

<sup>1</sup>Language selection is discussed in Chapter 8 of this manual.

Handbook that describes the PDP-11 computer you are using, and a User's Guide or Maintenance Manual for each peripheral device in your computer system. These manuals tell you how to operate the devices and give you special programming information that you may need if you intend to write device drivers or special system software involving the devices.

Software manuals<sup>1</sup> describe the operating system and the language processors. RT-11 software documentation falls into three major categories: introductory or once-only manuals (intended to be used once and then stored away); console manuals (intended to be used at the computer); and desk/console manuals (intended to be used at your desk for reference purposes).

## Software Manuals

Once-only manuals include this manual and others that are needed only when your system is initially installed. You may have little or no occasion to use these manuals once your computer system is in operation and you are familiar with its use.

Console manuals are those manuals that tell you how to use the computer system. They describe in detail command usage and syntax, list summaries of system operations, and give the meanings of system messages. The *RT-11 System User's Guide* is an example of a console manual.

Desk/console manuals are those manuals that you continually use for reference as you write your own application programs. These manuals include the general language reference manuals and the advanced programming manuals that contain programming information specific to the RT-11 computer system. The *RT-11 Software Support Manual* is an example of a desk/console manual.

Source listings are actual listings of the assembly language code that makes up the RT-11 operating system. These listings are very detailed and are generally needed only if you intend to modify the system software. They can be ordered on microfiche from the DIGITAL Software Distribution Center.

## Source Listings

This completes a general introduction to the RT-11 computer system. Subsequent chapters of this manual describe how you use the various system components mentioned here to perform a series of related computer operations. You begin in Chapter 2 by learning how to start the RT-11 computer system.

---

<sup>1</sup>All RT-11-related software manuals are listed and described in the *Guide to RT-11 Documentation*. Many of these manuals are provided with your system; others can be ordered from the DIGITAL Software Distribution Center.

## REFERENCES

*Digital Equipment Corporation Reference Service, Volume 2: Products and Services.* Maynard, Mass.: Digital Equipment Corporation, 1982.

An overview of the PDP-11 family products and services; includes capsule descriptions of the various PDP-11 computers, peripherals, and operating systems, and describes the supportive services provided by DIGITAL.

Eckhouse, Richard H. and Morris, L. Robert, *Minicomputer Systems: Organization, Programming, and Applications (PDP-11)*. Englewood Cliffs, N.J.: Prentice-Hall, 1979.

A guide to programming fundamentals, PDP-11 organization and structure, and programming techniques. See Chapters 1, 2, and 3.

*Guide to RT-11 Documentation (AA-5285G-TC)*. Maynard, Mass.: Digital Equipment Corporation, 1983.

A listing and brief summary of current RT-11-related software documentation.

Katzan, Harry Jr., Information Technology, *The Human Use of Computers*. New York: Mason & Lipscomb, Petrocelli Books, 1974.

An introductory textbook covering basic computing concepts, programming languages, and topics in computers and society. See Chapters 1, 2, 4, 5, and 10.

*PDP-11 Peripherals Handbook*. Maynard, Mass.: Digital Equipment Corporation, 1981-82.

A technical summary of the PDP-11 peripheral devices; includes descriptions, specifications, programming, and interfacing information for PDP-11 peripheral devices.

*PDP-11 Processor Handbook*. Maynard, Mass.: Digital Equipment Corporation, 1981.

A hardware manual for the owners and users of the PDP-11 family of computers and for those who will be using the PDP-11 assembly language instruction set.

*PDP-11 Software Handbook (EB-21759-20)*. Maynard, Mass.: Digital Equipment Corporation, 1982-83.

A general overview and introduction to available PDP-11 software, operating systems, and language processors.

Spencer, Donald D., *Fundamentals of Digital Computers*. Indianapolis, Kansas City, New York: Howard W. Sams, Bobbs-Merrill, 1969.

A discussion of the history and evolution of computers, computer applications, and fundamentals of computer use. See Chapters 1 through 12 and Chapter 20.

## CHAPTER 2 STARTING THE RT-11 COMPUTER SYSTEM

Before you can use the RT-11 computer system to perform any operations, you must start it. Starting the system involves turning on the computer and the various hardware devices and loading the appropriate software components into computer memory.

Within every PDP-11 computer is a physical, designated storage area called memory. Computer memory is where system information and data are temporarily loaded and stored for use during the various system operations.

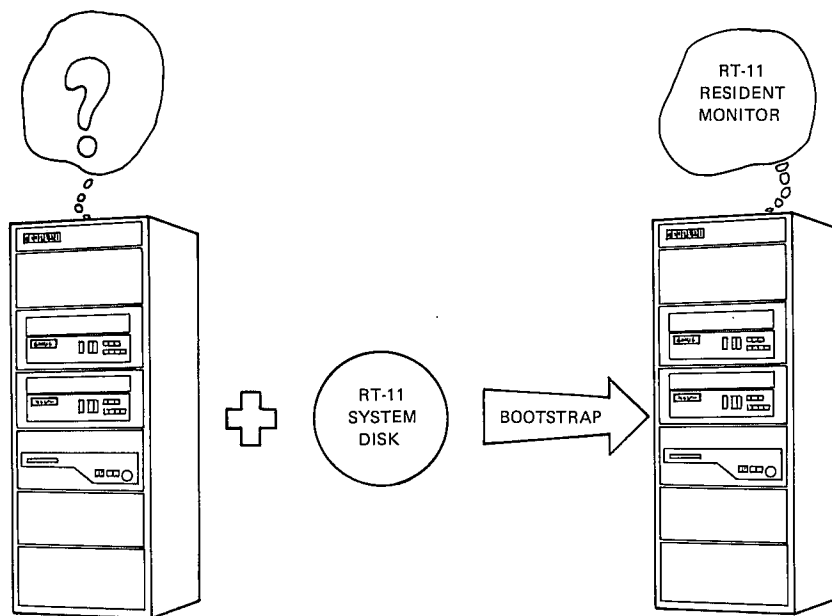
### COMPUTER MEMORY

Each time you use the computer system, there may already be information in computer memory, left by the person who used the system last. For example, there may be the results or data of another user's program; there may be the results of a particular system operation; there may even be an entirely different operating system in memory. For your purposes, computer memory must contain the RT-11 operating system, and specifically the RT-11 monitor program. Thus, your first operation as a system user is to transfer the monitor program from the disk device, where it was stored during system installation, to computer memory, where you can use it. The process of transferring the RT-11 monitor to memory is called bootstrapping the system; it is the only system operation that requires you to use the operator's console on the front panel of the computer (see Figure 2-1).

Starting the RT-11 computer system requires that you know how to operate your system's hardware devices. Since you may not have had the opportunity to use any of the devices yet, ask an experienced user to help you the first time. Follow the instructions in the section in this chapter entitled "Bootstrap Procedure." If necessary, refer to the *RT-11 Automatic Installation Booklet*, the *RT-11 Installation Guide*, or the various hardware manuals provided with your system.

### HARDWARE CONFIGURATION

First read through the following material and fill in the appropriate information where requested. You should be able to determine all responses by checking the *RT-11 Automatic Installation Booklet* or the *RT-11 Installation Guide*.



**Figure 2-1 Bootstrap/Computer Relationship**

**NOTE**

If your system device is a diskette, you need to build four volumes and, when running some of the demonstration programs, limit the volumes to the components needed to execute the programs. Also, you need to preserve the distribution volume you received from DIGITAL by making backup copies. The RT-11 automatic installation procedure performs these functions for you. If you did not use the automatic installation procedure to install your RT-11 system, the *RT-11 Installation Guide* will provide you with the commands you need to copy and preserve the distribution volume and create the volumes for use with this manual.

You must have the following materials to start the system and to perform the exercises in this manual:

- The volume containing the RT-11 operating system (called the system volume); refer to Section 2.3.6 of the *RT-11 Installation Guide* for the list of components you will need on your system volume to perform the exercises
- The volume containing the FORTRAN IV and/or BASIC-11 language processors if these languages are not stored on the system volume (available only to FORTRAN IV and BASIC-11 users)

- A volume for program storage (for example, magtape or another disk or diskette); this volume should contain no important information since all information on it will be erased during a later computer exercise
- A copy of the *RT-11 Automatic Installation Booklet* or the *RT-11 Installation Guide*

NOTE

You can find hardware configuration information in the various hardware manuals provided with your system. Instructions for starting (bootstrapping) your RT-11 system appear in the *RT-11 Automatic Installation Booklet* and the *RT-11 Installation Guide*. This information should be adequate for you to answer all the questions asked here. If you have trouble, see Appendix B, *Suggestions for Bootstrapping the System*. Do not continue to any other chapter in this manual until you understand the following configuration information and can bootstrap the system yourself.

1. What kind of terminal device are you using (for example, LA120 DECwriter III, VT100 video terminal)?
2. Is your computer a PDP-11/23-PLUS, PDP-11/24, or PDP-11/44?
3. Does your computer operator's console have pushbuttons or switches?
4. How much memory does your computer have?

**Terminal**

**Computer**

5. What kind of system volume are you using (for example, RL02 disk, RX02 diskette)?
6. What is the two-letter mnemonic for this volume (typical mnemonics are given in Table 2-1; respond with the mnemonic for your own volume)?

**System Volume**

**Table 2-1 Representative System Volumes**

Volume	Mnemonic
RD50/RD51 Disk (PC350)	DW
RX01 Diskette	DX
RX02 Diskette	DY
RX50 Diskette (PC325/PC350)	DZ
RK05 Disk	RK*
RK06/07 Disk	DM
RC25/RD51 Disk, RX50 Diskette	DU
RL01/02 Disk	DL

\*Use DK to bootstrap from an RK05 disk.

**Storage Volume**

7. What volume are you using for program storage (for example, TS11 magtape, RL02 disk)?
8. In which device unit will you use this volume (choose any available device unit — for example, 0, 1)?

**Optional Devices and Supported Languages**

9. What peripheral devices are part of your system (for example, line printer, magtape, VT11 display hardware; list all devices other than the terminal and the computer)?
10. What programming languages does your system support (MACRO-11 or BASIC-11, for example)?

**BOOTSTRAP PROCEDURE**

Once you have determined your hardware configuration, you are ready to bootstrap the system. The purpose of the bootstrap procedure is to load and start the RT-11 monitor in computer memory, thus activating the RT-11 computer system for your use.

**NOTE**

If your answer to question 2 in the Hardware Configuration section is YES, continue to the next paragraph. Otherwise, read the section entitled Bootstrapping the System, in Appendix A, for bootstrap instructions.

The bootstrapping procedure for the RT-11 computer system on a PDP-11/23-PLUS, PDP-11/24, or PDP-11/44 processor consists of the following steps. For more detailed instructions on

the bootstrap operation, refer to the *RT-11 Automatic Installation Booklet*.

1. Turn the terminal to an on-line condition.
2. Make sure that the computer power is on and that the computer is not already in use.
  - If your computer is a PDP-11/23-PLUS, power up the system by lifting the AUX toggle switch to the ON position. The red PWR OK indicator on the front panel will light up if the system was successfully turned on.
  - If your computer is a PDP-11/24 or a PDP-11/44, power up the system by turning the status selector key to the LOCAL position. The red DC ON indicator on the front panel will light up if the system was successfully turned on.
3. Stop the computer.
  - If your computer is a PDP-11/23-PLUS, lift the HALT toggle switch to the up position. The AUX toggle switch that you lifted in the previous step and the HALT toggle switch can be lifted simultaneously.
  - If your computer is a PDP-11/24 or a PDP-11/44, push the HALT/CONT/BOOT horizontal toggle switch to the HALT position.
4. Load the system volume in its corresponding device unit 0. Make sure that the system volume is write-protected (for all except RX01 or RX02 diskettes, which are always write-enabled).
5. Load the storage volume in the device unit noted in question 8 in the Hardware Configuration section. Make sure that this volume is write-enabled.
6. Boot the system.
  - If your computer is a PDP-11/23-PLUS, lift the RESTART toggle switch on the front control panel. This switch will not remain in the up position; it will spring back to the center position.
  - If your computer is a PDP-11/24 or a PDP-11/44, push the HALT/CONT/BOOT horizontal toggle switch to the BOOT position. This switch will not remain in the right-most position; it will spring back to the center position.

The red RUN indicator on the front control panel should now be illuminated.

A series of self-diagnostic routines to check out the system are then executed. The execution of these routines may take up to a minute, depending upon how much memory is installed in your system. A prompt appears on your console terminal when execution of the routines is completed. The prompt that appears is dependent upon the type of processor you are using. Table 2-2 provides the prompts that appear and the corresponding responses which must be supplied if you are using the PDP-11/23-PLUS, PDP-11/24, or PDP-11/44 processor.

**Table 2-2 Bootstrap Prompts and Responses**

Processor	Prompt	Response
PDP-11/23-PLUS	TESTING MEMORY www.KW START?	dd[n](RET)
PDP-11/24	bbbbbbbb	<none>
PDP-11/44	>>>	B dd[n](RET)

www. = amount of memory in K-words (decimal)  
 bbbbbbbb. = amount of memory in K-bytes (octal)  
 dd[n] = device mnemonic (dd) and unit number (n)

- Respond to the prompt that appears on your console terminal by typing the appropriate response (refer to Table 2-2) followed by a carriage return.

**NOTE**

Refer to question 6 in the Hardware Configuration section for the two-letter device mnemonic (dd) and refer to question 8 for the device unit number (n).

You should now direct your attention to the console terminal, since system interaction continues on this device.

**REFERENCES**

*PDP-11 Processor Handbook*, Maynard, Mass.: Digital Equipment Corporation, 1981.

A hardware manual for the owners and users of the PDP-11 family of computers and for those who will be using the PDP-11 assembly language instruction set.

*RT-11 Automatic Installation Booklet: RX02 Diskettes* (AA-M235A-TC),  
*RT-11 Automatic Installation Booklet: RL02 Disk* (AA-M236A-TC),  
*RT-11 Automatic Installation Booklet: RC25 Disk* (AA-M237A-TC), and  
*RT-11 Automatic Installation Booklet: MICRO/PDP-11* (AA-M238A-TC).  
Maynard, Mass.: Digital Equipment Corporation, 1983.

RT-11-specific software booklets which provide basic instructions for using the automatic installation process to install and test the RT-11 monitors, system programs, and certain languages.

*RT-11 Installation Guide* (AA-H376B-TC) and *RT-11 System Release Notes* (AA-5286E-TC). Maynard, Mass.: Digital Equipment Corporation, 1983.

Two RT-11-specific software manuals that contain instructions for installing, customizing, and starting the RT-11 computer system.

*RX8/RX11 Floppy Disk System Maintenance Manual* (EK-ORX01-MM-PRE2). Maynard, Mass.: Digital Equipment Corporation, 1975.

A hardware manual for the owners and operators of RX01 diskettes and for those who will be programming computers to interact with these devices.

*VT100 User Guide* (EK-VT100-UG-002). Maynard, Mass.: Digital Equipment Corporation, 1978.

A manual for the owners and operators of the VT100 video terminal and for those who will be programming computers to interact with these devices.



## CHAPTER 3 INTERACTING WITH THE RT-11 COMPUTER SYSTEM

Interaction with the RT-11 computer system involves an exchange of information between you (the user) and the software operating system. The exchange may be active, with you dictating command information from the terminal keyboard and the system responding immediately; or it may involve the storing of information on mass storage volumes for later use.

During the bootstrap procedure you activated the RT-11 computer system by loading and starting the monitor program in computer memory. One of the functions of the monitor program is to provide you with the capability to use the console terminal. Since the console terminal can perform both input and output operations, it is used to interface between the system and the user. With it, you can:

- Type the commands that control system operation
- Receive messages and responses from the system

All console terminals have a keyboard — used to enter information — and a paper output device or video screen — used to echo characters typed at the keyboard and to print system messages and responses. Figure 3-1 shows two commonly used terminals, the LA120 and the VT100.

These two terminals differ in their output mechanism. While the LA120 terminal has a paper printer, the VT100 has a video screen. The paper printer and the screen serve the same purpose — they show user input and system responses; however, paper output can be saved for later use while screen output is temporary. The keyboards of both terminals are shown in Figure 3-2.

### USING THE CONSOLE TERMINAL TO EXCHANGE INFORMATION

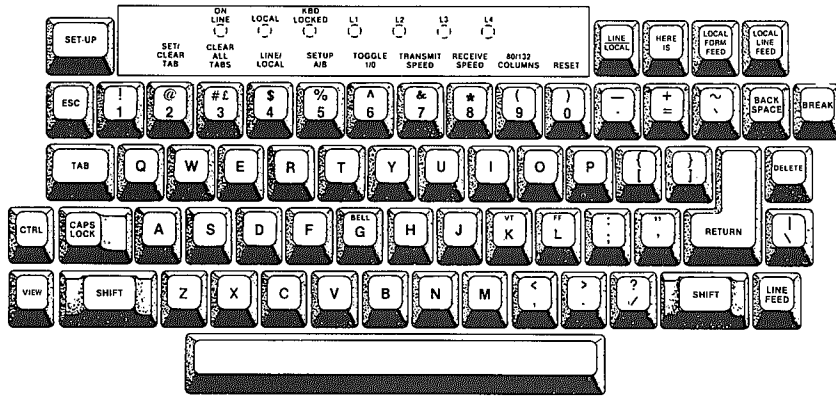


**LA120**

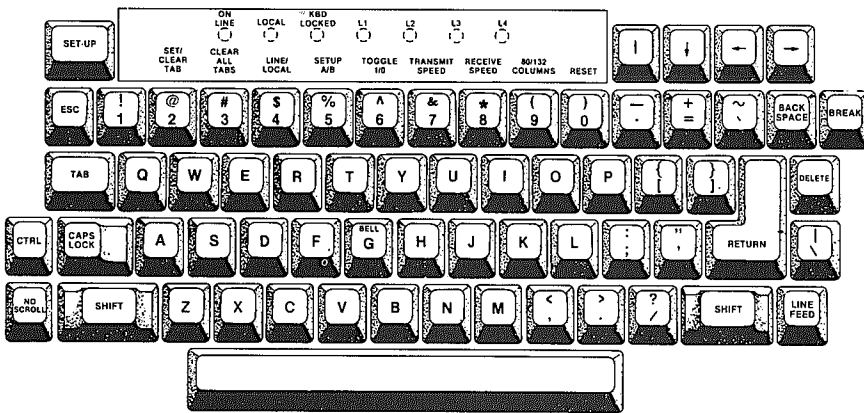


**VT100**

**Figure 3-1 LA120/VT100 Terminals**



LA120



VT100

Figure 3-2 LA120/VT100 Keyboard Layouts

Using Figure 3-2 as a guide, study your own terminal keyboard. First, notice that the keys for the alphabetic characters are positioned in the same way as on most standard typewriters. The SHIFT key allows you to select between numeric and special characters and between uppercase and lowercase characters.<sup>1</sup> The position of the numeric and special characters varies somewhat among the different terminals, so you may need to hunt for a particular key until you become familiar with your own terminal.

Locate the DELETE key. This key is used to correct a typing mistake. Pressing the key once cancels the last character typed. Pressing it twice cancels the last two characters, and so on, back to the beginning of the line.

<sup>1</sup>With the exception of system messages and one other exception explained in Chapter 5, the RT-11 computer system uses uppercase characters exclusively.

Locate the TAB key. Tab stops on a computer terminal are positioned every eight spaces across the line, beginning at column 1. Pressing the TAB key moves the character pointer (that is, the position on the line where the next character will be typed) to the beginning of the next tab stop.

The key marked RETURN performs a carriage return; it both returns the character pointer to the beginning of the line and advances it to the next line. This key is used to terminate the line currently being typed and to terminate certain RT-11 system commands.

Locate the ESC key and the LINE FEED key. These are special command terminators that are described in Chapters 5 and 14.

An important key is the CTRL key. It is always used with another character key to perform one of several system operations. CTRL commands are explained in detail when you begin to use them later in the manual.

Table 3-1 reviews the console terminal keyboard characters. Keys not mentioned are not used by the RT-11 computer system and can be ignored.

You will have an opportunity to become familiar with your terminal keyboard as you perform the demonstrations in this manual.

The console terminal also displays messages and responses from the system. These messages and responses provide or request information. Error messages are an example of informational output; they help you detect typing errors, programming errors, and system malfunctions. If an error message appears on your console terminal while you are performing the demonstrations in this manual, refer to the *RT-11 System Message Manual* for an explanation of the cause of the message and a description of the corrective action that should be taken.

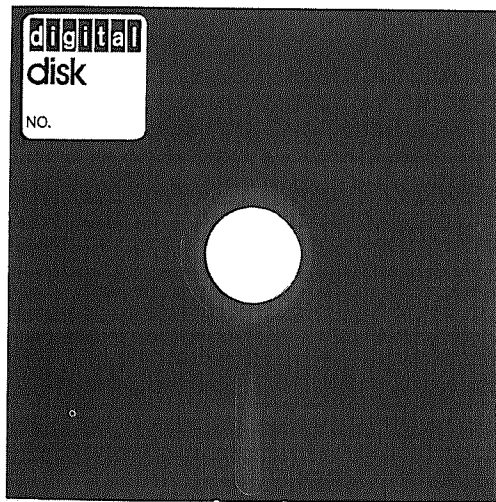
## **USING MASS STORAGE VOLUMES**

Mass storage volumes provide an area (apart from computer memory) to keep information for later use. The information may be user application programs, data needed by a program, the results of a program run, textual information, batch-type programs, and so on. As an example, the RT-11 operating system is stored on a mass storage volume called the system volume. When information is needed, as it was during bootstrapping, information from the storage volume is transferred into computer memory.

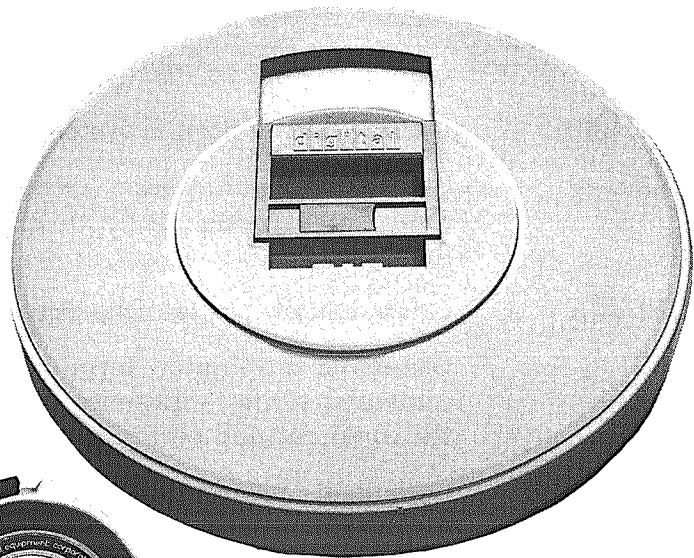
Before you can access the information stored on a storage volume, however, you must first insert the volume (the medium) into its corresponding device unit (drive), the hardware device connected to the computer. Once a volume has been inserted into a device unit, the device unit's symbol identifies the volume. There may be more than one device unit for a volume, each individual device unit is numbered 0, 1, 2, and so on. As you learned in the bootstrap procedure, the system volume is inserted in device unit 0 and remains in it as long as you are using the system. Other storage volumes can be inserted in any available device units. Figure 3-3 illustrates several mass storage volumes.

**Table 3-1 Keyboard Characters**

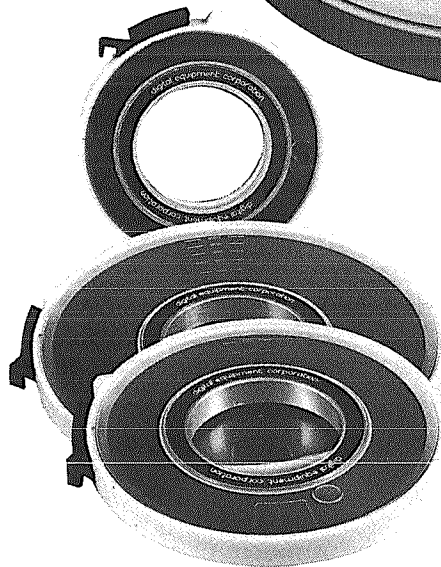
<b>Key</b>	<b>Function</b>
BACK SPACE	Ignored during normal system use
BREAK	Ignored during normal system use
CTRL	Control; part of several two-key command combinations that perform specific system functions
DELETE	Erase; cancels the last character typed
ESC	Command terminator; terminates an editing command string; transmits the command to the computer and performs a carriage return
LINE FEED	Command terminator; terminates certain system commands; transmits the command to the computer and performs a carriage return
REPEAT	Ignored during normal system use
RETURN	Line terminator, command terminator; terminates the current line; terminates certain system commands; transmits the command to the computer and performs a carriage return
SHIFT	Selects the uppermost of two characters appearing on a key
TAB	Moves the character pointer ahead to the beginning of the next tab stop
any other key	Transmits the alphanumeric or special character to the computer



**Diskette**



**RL02**



**Magtape**

**Figure 3-3 Mass Storage Volumes**

Mass storage volumes hold large amounts of information. Most volumes, however, are physically small enough so that you can transport them from the system, to your desk perhaps, or to another computer system. In addition to disks (shown earlier in Figure 1-4), magtapes are also mass storage volumes.

You store information on a mass storage volume in the form of files. Each file is a logical collection of data. Files may be parts of programs or entire programs, program input data, or text, such as a letter or report. Whatever its content, each file is treated as a unit and occupies a fixed area of the volume.

Every file on a mass storage volume has a unique name that is composed of a file name and a file type. The file name and file type identify the file and distinguish it from other files on the volume. You can instruct the system to print on your terminal the names of all files on a volume. The resulting list is called the volume directory listing. By referring to the volume directory, you can find the name, size, and creation date of each file on that volume and delete old files that you no longer need. Whenever you perform an operation that affects the contents of the volume, a new volume directory reflects the change.

## **File Storage**

Occasionally, after many files are added to a storage volume, the volume has no room for new information. A storage volume may also become damaged, lost, stolen, or worn through use. For these reasons it is a good idea to have several extra storage volumes on hand and to protect your more important files against accidental erasure or loss.

One way to protect a file is to make a copy of it on a second storage volume. The copy, called a backup file, insures you against the loss or damage of your original file (or its respective storage volume).

Some storage volumes provide a mechanism that protects files against accidental erasure. This mechanism is generally a switch on the volume itself, or on the device unit, that you can set to a write-protect or write-enable condition (as you did during bootstrapping). When the volume is write-protected, information can be copied only from the volume to computer memory or to another volume that is write-enabled. A volume that is write-enabled, on the other hand, also allows information to be copied from memory back to the volume.

## **File Protection**

The RT-11 operating system itself provides a protection feature. This optional feature requires that you confirm certain system commands that might otherwise erase important information. The system also issues prompting messages so that you provide the proper file information when it is needed by a command.

Chapter 4 and succeeding chapters require you to use the terminal to enter command information and perform file copy and other system operations. Before you continue, make sure that there is a backup copy of your system volume. If you cannot locate one, read Appendix B, Backing Up the System Volume, before going on.

## **REFERENCES**

*RT-11 System Message Manual* (AA-5284D-TC). Maynard, Mass.: Digital Equipment Corporation, 1983.

An explanation of system messages that may occur during normal system use; includes required user actions.

*VT100 User Guide* (EK-VT100-UG-002). Maynard, Mass.: Digital Equipment Corporation, 1978.

A manual for the owners and operators of the VT100 video terminal and for those who will be programming computers to interact with these devices.

## CHAPTER 4 USING THE MONITOR COMMAND LANGUAGE

During the bootstrap operation, the RT-11 monitor was copied into computer memory and started. The RT-11 monitor is actually many different components working together to supply basic system functions. For example, the part of the monitor called the resident monitor (RMON) provides the console terminal service and central program code necessary for a working environment for both system and user programs. The resident monitor is so named because it always remains in computer memory, regardless of other system operations that may be occurring. Other parts of the monitor are brought into memory from the system volume as needed. These include the user service routine (USR), which provides support for the RT-11 file system, and the keyboard monitor (KMON), which controls terminal keyboard interaction. From your standpoint, the keyboard monitor is the most visible part of the system software. Among other services, it supplies the monitor command language that you use to control system operations.

The monitor command language is a set of English-like command words that you type on the terminal keyboard to initiate and control system operations. You can type a command in one of two general formats: a long format or a short format. The long format causes the system to print prompting messages. These messages ask you to supply specific information, such as file names and device names. The long format is helpful until you become familiar with the commands. You will then probably prefer to use the short format, which allows you to enter all required information on a single command line and provides prompts only if you do not supply necessary information. Both formats are demonstrated throughout this manual.

You terminate all monitor commands with a carriage return. That is, after you type the required command information, you press the carriage return key (represented in this manual by **RET**). This instructs the monitor to initiate the command and to perform the operation.

A prompt character — a period at the left margin of the terminal printer or screen — appears whenever the RT-11 monitor is waiting for you to type a command. The period is your cue that the system is in the monitor command mode and ready to accept a monitor command. Check the output on your terminal printer or screen. You should see the following at the left margin:

```
RT-11FB    V05.xx
```

### ENTERING COMMAND INFORMATION

RT-11FB identifies the RT-11 monitor called the foreground/background (FB) monitor. Following this is the version (and update) number of the system in use, in this case, Version 5. The period on the next line indicates that the system is in the monitor command mode and is waiting for you to type a monitor command.

## General Command Format

Whenever you issue a monitor command, you must supply certain information to guide command processing. This information includes the following (square brackets indicate optional qualifiers and characters):

COMMAND[/option] First you indicate, by command, which system operation you want initiated. Command options are available to allow you to alter the normal (default) operation.

INPUT[/option] You next indicate, by device and file name, input information that is to be used during the operation. The system volume serves as the default input device. You must explicitly indicate other volumes that you want used for input, and you must usually indicate the file names and file types of the input files. Input file options are available to allow you to alter assumed (default) input operations.

OUTPUT[/option]<sup>1</sup> Finally you indicate, by device and file name, output information that is to be created as a result of the operation. The system volume serves as the default output device. You must explicitly indicate other volumes that you want used for output, and you must usually indicate the file names and file types of the output files to be created. Output file options are available to allow you to alter assumed (default) output operations.

As mentioned earlier, you can type this command information on the terminal keyboard in one of two formats; illustrations of both follow:

---

<sup>1</sup>OUTPUT[/option] is not always used; sometimes output must be specified as COMMAND[/option] INPUT/OUTPUT:filespec.

**Long Command Format** (system prompts for specific information)

```
.COMMAND[/option]␣  
INPUT PROMPT? INPUT[/option]␣  
OUTPUT PROMPT? OUTPUT[/option]␣
```

**Short Command Format** (no prompts)

```
.COMMAND[/option] INPUT[/option] OUTPUT[/option]␣
```

Notice that a slash character (/) separates an option from the portion of the command that it qualifies, and a carriage return ␣ terminates each individual command line. When you have supplied all the necessary information, the carriage return signals the monitor to execute the command. You can use either format; both are demonstrated throughout this manual.

In addition to monitor commands, RT-11 has several special function commands, called control commands, that you type by first pressing and holding down the CTRL key on the terminal keyboard and then typing the letter key of the command. To execute the CTRL/C command, for example, type the letter C while holding down the CTRL key. These control commands require no terminator; the system performs the function as soon as you type the command.

Control commands are used to interrupt program execution, to inhibit terminal output, and to perform other similar special system operations. They are described in the manual as you need to use them.

During the course of this chapter, and throughout the remainder of the manual, you will use a number of monitor commands to perform some common system operations. For example, you will list the directories of device volumes, copy files between devices, create files, and execute system and user programs. You perform these operations by re-creating on the terminal keyboard the examples already provided for you.

You should first read the entire explanation of a command to be aware of its format, the operation it performs, and the options that are available. Then type the command on the terminal keyboard exactly as you see it used. Characters that you type appear in the demonstrations in red print. Characters that are system responses are shown in black print.

Table 4-1 lists symbols that you will see used throughout the demonstrations. These symbols represent various keys on the

## Control Commands

## Re-Creating the Examples

terminal keyboard. When you see one of these symbols in a command line, type the appropriate key on the keyboard.

Table 4-1 Keyboard Symbols

Symbol	Type
<code>(RET)</code>	carriage return key
<code>(LF)</code>	line feed key
<code>(SP)</code>	space bar (once for each time the symbol is shown). Assume that you should type a single space unless you are otherwise instructed; the space symbol is used only if there is doubt about the number of spaces to type.
<code>(TAB)</code>	TAB key (once for each time the symbol is shown)
<code>(DEL)</code>	DELETE key (once for each time the symbol is shown)
<code>(ESC)</code>	ESCAPE key (once for each time the symbol is shown)
<code>(CTRL/x)</code>	CTRL key (hold down CTRL key while typing the letter character [x])

**CORRECTING  
TYPING  
MISTAKES**

**CTRL/U**

All commands that you give the system are typed on the terminal keyboard. If you make a mistake while typing a command, you can correct it in one of two ways.

One way to correct a typing error is to use the DELETE key on the keyboard. Pressing the DELETE key once cancels the character just typed; pressing it a second time cancels the next to last character typed, and so on, from right to left, until the beginning of the line is reached. Then additional DELETES are ignored.

The second way to correct a typing error is to use CTRL/U, a special control command. Typing this command once is equivalent to typing as many DELETES as are needed to cancel every character in the current line.

Type on the keyboard the letters DABE, followed by two DELETES, followed by the letters TE, and notice the system's response:

,DABE (DEL) (DEL) TE

The monitor echoes each deleted character and encloses them within backslashes. As far as the monitor is concerned, the only characters you have typed are DATE.

.DABE\EB\TE

Thus, your current line is DATE. Continue by typing a CTRL/U. Remember to first press and hold down the CTRL key and then type the U key; no carriage return is necessary.

**CTRL/U**

Notice that CTRL/U echoes on the terminal printer or screen as ^U.

.DABE\EB\TE^U

All characters on the line are canceled, and the character pointer is moved to the beginning of a new line so that you can enter another command. You are still in the monitor command mode even though no prompting period appears at the left margin.

Once the carriage return or line feed key is pressed, the previous line cannot be corrected with DELETE or CTRL/U.

These two methods are commonly used to correct typing errors made at the keyboard. You can choose whichever method seems most convenient.

The kinds of command operations that you usually perform immediately after the monitor is bootstrapped are those that set up initial conditions, such as the current date and time of day, and those that initialize and prepare the system for future operations such as file transfers. If your system has VT11 display hardware that you want to use, you should also enable (turn on) the graphics display screen.

## INITIAL MONITOR COMMAND OPERATIONS

Display hardware on an RT-11 computer system consists of a cathode ray tube that allows programs to use graphics displays. If your system has display hardware<sup>1</sup> (Figure 4-1), you can use the graphics screen in place of the terminal printer or screen.

## Using VT11 Display Hardware

### NOTE

Check question 9 in the Hardware Configuration section of Chapter 2 to determine if your system has display hardware. If you do not have display hardware, go on to the next section, Entering the Date and Time-of-Day.

---

<sup>1</sup>Video terminal screens are not considered graphics display hardware.

GT

The monitor command that enables the graphics screen is the GT command. The GT command is used to change the condition of the graphics display. In this case, you will use it to activate the graphics display hardware so that the VT11 display screen replaces the console terminal printer or screen as the terminal output device.

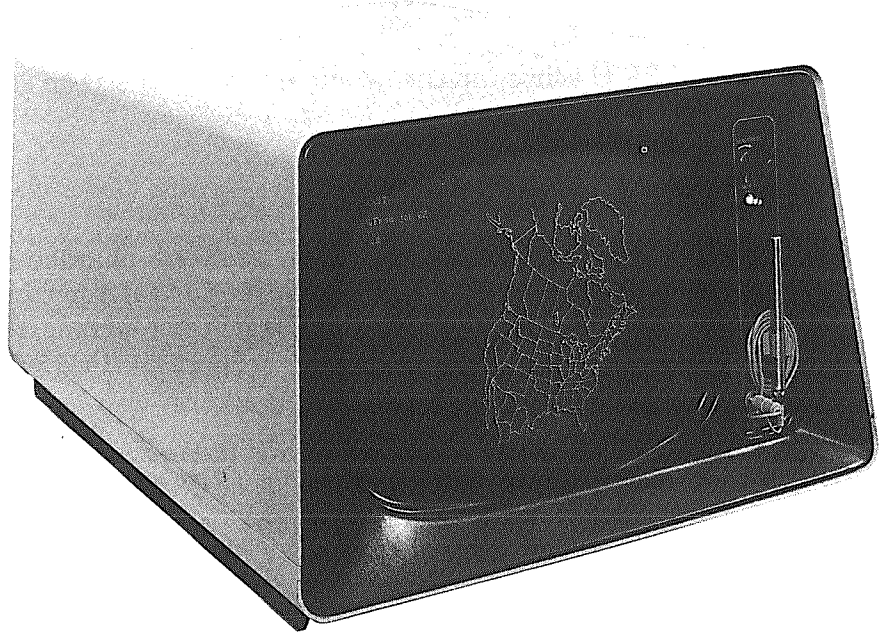


Figure 4-1 VT11 Display Hardware

Type the following on your terminal keyboard (if necessary, refer to Table 4-1 to review the special symbols):

### Long and Short Command Format

```
.GT ON(RET)
```

If your system does not have display hardware, the monitor prints a message<sup>1</sup> on the terminal printer or screen informing you that the command is invalid for your system configuration:

```
?KMON-F-Invalid command
```

Otherwise, the command is accepted. You should notice that all character-echoing and system responses are now directed to the graphics screen instead of to the terminal printer or screen. After the command has been accepted, a period appears on the graphics screen, indicating that the system is waiting for another command. The character pointer is visible as a blinking rectangular cursor situated after the period. (In the edit mode, the cursor is L-shaped.)

<sup>1</sup>The meanings of all system messages are listed in the *RT-11 System Message Manual*.

Like output on the terminal screen, output that appears on the graphics screen is temporary. Once the screen is filled, lines are rolled off the top and are lost to view. However, if your terminal has a printer, a special control command allows you to control console terminal output so that it appears on both the graphics screen and the terminal printer simultaneously. In this manner, you can direct selected portions of terminal output — directory listings, for example — to be both displayed and printed at the same time. The advantage of this is that although the display copy is eventually lost, you have a printed copy for later use.

The control command that provides this function is CTRL/E, which is initiated by holding the CTRL key down while typing the E key. No carriage return is necessary. When you type this command, no characters echo on the graphics screen, but you should notice that all subsequent characters (both input and output) appear on both the graphics screen and the terminal printer.

CTRL/E

Thus, if your terminal has a printer and you wish to use the printer in addition to your VT11 graphics screen, type once:

`CTRL/E` (Remember, this command does not echo.)

Now type the following and notice where the characters echo:

`•WRONG COMMAND CTRL/U`

To disable the printer at any time so that character echoing occurs only on the graphics screen, type another CTRL/E command:

`CTRL/E`

Finally, to return terminal output control to the terminal, disabling the graphics screen, use the GT OFF command; this changes the terminal device handler back to its original output setting:

### Long and Short Command Format

`•GT OFF RET`

Decide now whether to use the graphics screen for the remaining demonstrations. If so, use the GT ON command to enable the graphics screen, and remember that the CTRL/E command is available when you wish to produce simultaneous output.

## Entering the Date and Time-of-Day

DATE

Entering the current date and time-of-day helps in record-keeping for system operations. Later, you can identify when system operations were performed.

For example, by entering the current date you instruct the system to assign this date to all files you create. The date will also appear in volume directories and listings produced by the various language processors and utility programs. If your system has a clock, by specifying the current time-of-day you instruct the system to keep track of time based on the time you set. The current time is printed on listings when they are produced, and may also be used to control certain program operations.

Enter the date by typing the monitor DATE command with the day, month, and year as follows (there is only one format):

### Long and Short Command Format

```
,DATE 8-JAN-83Ⓡ
```

This sets the date to January 8, 1983. Since this date is not current, reenter the correct date using the same command format:

```
,DATE dd-mmm-yyⓇ
```

Typing the new date overrides the previous date.

The date that is set is temporary. You must reenter it whenever you bootstrap the system.

TIME

The monitor TIME command is used to set the time-of-day, specified in 24-hour notation. The system keeps track of time in hours, minutes, and seconds, based on the initial time that you enter in the command. Enter the time as follows (there is only one format):

### Long and Short Command Format

```
,TIME 15:01:00Ⓡ
```

If your system does not have a clock, the monitor prints a message on the terminal; this message informs you that the command is not valid for your system configuration:

```
?KMON-W-No clock
```

Otherwise, the time is set to 3:01 p.m. If your system has a clock, reenter the correct time, using the same command format:

```
,TIME hh:mm:ssⓇ
```

Typing the new time overrides the previous time.

The system's clock stops when the system stops running. If you want the time to be kept current, you must reenter it whenever you bootstrap the system. If your system has a clock and you do not set the time, the TIME command will return the time elapsed since the last hardware boot.

To check the time or date at any time while you are using the system, simply type either the DATE command or the TIME command, followed by a carriage return only:

#### Long and Short Command Format

```
. DATE(RET)
8-JAN-83
. TIME(RET)
15:06:19
```

The system responds by printing the date or the time, based on the information you previously entered. If the system responds to the DATE command with the message *?KMON-W-No date*, the date has not been set since the system was last bootstrapped.

Each hardware device in the RT-11 system is identified by a two-letter mnemonic. The mnemonics, listed in Table 4-2, are defined in the system software and are recognized and used by the operating system. These are the device names that you generally use in command input and output lines. However, you may want to change any of these device names temporarily, for a variety of reasons. The following paragraphs describe both using the physical device names shown in Table 4-2 and assigning logical (temporary) device names to devices.

#### Assigning Logical Names to Devices

Table 4-2 Physical Device Names

Mnemonic	Device
DUn:	RC25/RD51 Disk, RX50 Diskette
DLn:	RL01/02 Disk
DMn:	RK06/07 Disk
DW:	RD50/RD51 Disk (PC350)
DXn:	RX01 Diskette
DYn:	RX02 Diskette
DZn:	RX50 Diskette (PC325/PC350)
LP:	Line Printer
LS:	Serial Line Printer
MMn:	TJU16 Magtape
MSn:	TS11 Magtape
MTn:	TM11 Magtape
RKn:	RK05/RK11 Disk
TT:	Console Terminal

Two additional logical device names are used. These special names are described in Table 4-3.

Table 4-3 Special Logical Device Names

Mnemonic	Device
SY:	The volume from which the monitor was bootstrapped; that is, the system volume.
DK:	The default storage volume (initially the same as SY:; that is, the system volume).

You use device names in the input and output portions of a command line to identify where input information can be found and where output information will be sent. If a file is involved, you also include its file name and file type, in the following format:

```
devicename:filename.filetype
```

The device name is followed by a colon and is always separated from any file name and file type by a colon. The device name is generally one of the mnemonics listed in Tables 4-2 and 4-3. When you use a device name in any command, you must also include the device unit number (represented by the letter *n* in Table 4-2) unless the number is 0. The system assumes unit 0 of the device if no unit number is given. Thus, diskette unit 0 is DY: or DY0:; diskette unit 1 is DY1:; RK: disk unit 2 is RK2:; and so on. Note that, according to Table 4-3, you can use the device mnemonic SY: or DK: for your system volume in addition to its standard device name. However, since the system volume is initially the default storage volume for all operations, you do not need to use a device name for your system volume.

The names listed in Tables 4-2 and 4-3 are the device names defined within the system software. However, you can change any of these name assignments temporarily, either by reassigning existing names to different devices or by assigning new logical names of your own choosing to devices.

You might want, for many reasons, to change a device name temporarily and assign it a logical name. You may, for example, have a program written for a device that is not available on your system. If you assign the program name to a device that is available, the program then uses that device instead.<sup>1</sup>

Since not all RT-11 users have access to the same kind of storage volume, you are instructed to assign the logical name VOL: to whatever volume you are using for storage. After you

<sup>1</sup>This is called device independence.

make this assignment, subsequent command lines can be the same for everyone using this manual.

Similarly, the special logical device name DK:, presently assigned to your system volume, could be assigned to any kind of storage volume. Not only would DK: signify your storage volume, regardless of its physical device name, but you could also avoid typing DK: since it is the default storage volume for most commands. (Only the R command requires that the file specified be on the system volume SY:.)

To assign a logical name to your storage volume, first determine its physical device name. Check questions 7 and 8 in the Hardware Configuration section of Chapter 2 to see which device and which device unit you are using for your storage volume. Translate this into the appropriate name and number using Table 4-2 as a guide.

Use the monitor ASSIGN command to change this physical name to a logical name. Substitute for physical-device-name in the following command lines the physical name and device unit number for your storage volume (for example, for RK05 disk unit 1, substitute RK1:).

ASSIGN

### Long Command Format

```
·ASSIGN(RET)
Physical device name? physical-device-name(RET)
Logical device name? VOL:(RET)
```

### Short Command Format

```
·ASSIGN physical-device-name VOL:(RET)
```

Once the assignment is made, the system recognizes the logical name VOL: as the device name for your storage volume. This is the only logical assignment you need to make. Since you are not changing the DK: assignment, the system volume remains the default device for all I/O operations.

As you continue to use the system, you may well make many device assignments and deassignments. To check the status of all assignments made during a computer session, you can use the monitor SHOW command to print on your terminal a list of all the logical assignments currently in effect. Use the SHOW command now to check the status of the assignment just made:

SHOW

### Long and Short Command Format

```
·SHOW(RET)
```

Check the list printed on your terminal to make sure that the code VOL: has been assigned to your storage volume. The letters VOL: should follow the appropriate device name in the list, as in the following response, in which VOL: represents disk unit 1:

```
TT
RK  (Resident)
    RK0 = SY , DK
    RK1 = VOL
LD
DX
DT
DD
CT
LP
LS
PC
BA
NL
13 free slots
```

Logical device assignments are temporary. Thus, if you want a logical device assignment to remain in effect, you must reassign it each time the system is bootstrapped.

## Listing Volume Directories

DIRECTORY

Both your system volume and your storage volume have directories, which are compiled lists of all the files stored on the volume. You can print a volume directory on your terminal, using the monitor DIRECTORY command.<sup>1</sup> To list the directory of your system volume, type:

### Long and Short Command Format

.DIRECTORY<sup>(R)</sup> (The system volume is the default device.)

CTRL/O

Since the directory of the system volume may be quite long, after approximately 10 lines have printed on the terminal, type:

<sup>(R)</sup>CTRL/O

This special control command, echoed as ^O, inhibits the remainder of the listing output from printing on the terminal, although the information on the total number of files and blocks is still given. When control returns to monitor command mode, look at the directory listing. At the top of the listing is today's date, as you entered it earlier in the DATE command. Following the date is a list of the files on the volume. Notice the two-column format of each line in the directory.

<sup>1</sup>Users of VT11 display hardware may wish to use the CTRL/E command to enable both the graphics screen and the terminal printer for the following exercises.

```

08-Jan-83
SWAP .SYS      26  26-Aug-82    RT11SJ.SYS    73  26-Aug-82
RT11FB.SYS    86  26-Aug-82    RT11BL.SYS    73  26-Aug-82
RT11XM.SYS    94  26-Aug-82    TT .SYS       2  26-Aug-82
DT .SYS       3  26-Aug-82    DP .SYS       3  26-Aug-82
DX .SYS       3  26-Aug-82    DY .SYS       4  26-Aug-82
RF .SYS       3  26-Aug-82    RK .SYS       3  26-Aug-82
DL .SYS       4  26-Aug-82    DU .SYS       4  26-Aug-82
DM .SYS       5  26-Aug-82    DS .SYS       3  26-Aug-82
DD .SYS       5  26-Aug-82    ^D

```

```

170 Files, 4264 Blocks
498 Free blocks

```

First the file name appears, followed by a dot and a file type that is frequently used to identify the file's format. For example, .SYS represents a system file; other RT-11 file types used to represent different kinds of files are listed in Table 4-4. After the file type is a number that indicates the size of the file. The size is given in blocks, a term used to designate a standard amount of information. A file that is 1 to 10 blocks long is fairly small, while a file over 100 blocks in length is quite large. The date on which the file was created is shown at the right. This space is empty if a date was not specified (with the DATE com-

**Table 4-4 File Types**

File Type	Meaning
.BAC	BASIC compiled file
.BAK	Editor backup file
.BAS	BASIC source file
.BAT	BATCH source file
.BUP	Backup/restore file
.COM	Indirect command file or IND indirect control file
.CTL	BATCH control file
.DAT	BASIC-11 or FORTRAN IV data file
.DBL	DIBOL source file
.DIF	SRCCOM output file
.DIR	Directory listing file
.DSK	Logical disk file
.FOR	FORTTRAN IV source file
.LOG	Batch log file
.LST	Listing file
.MAC	MACRO-11 source file
.MAP	Linker map file
.MLB	MACRO library file
.OBJ	MACRO-11, FORTRAN IV, or DIBOL object output file or library file
.REL	Executable foreground program file or system job
.SAV	Executable background program file
.SML	System MACRO library
.SYS	System files and handlers

mand) on the day the file was created. At the end, you are told how many files are on the volume, their total length, and the number of free blocks available for your use.

NOTE

Files furnished on the distribution medium have a protected status, which means they cannot be deleted. This is indicated by the letter P after the file size shown when you print a directory listing. You cannot perform any operation on a protected file if the result is to delete it. You can change the protected status of a file by using the RENAME keyboard monitor command with the /PROTECTION or /NOPROTECTION option; you can give a protected status to a file by using the PROTECT keyboard monitor command; and you can remove a protected status from a file by using the UNPROTECT keyboard monitor command (see the RT-11 System User's Guide).

DIRECTORY  
/BRIEF

You can also obtain an abbreviated directory, which omits file lengths and dates and lists only file names and file types in five-column format. To do this, you use the DIRECTORY command with its /BRIEF option. Type the following, and after several lines have listed, interrupt the directory by typing two CTRL/C command characters. This double control command echoes two ^Cs and requests the running program to abort immediately, regardless of what the program is doing (one CTRL/C aborts an executing program waiting for input from the console terminal). Control returns to monitor command mode.

CTRL/C CTRL/C

Long and Short Command Formats

```
,DIRECTORY/BRIEF(RET)
08-Jan-83
SWAP .SYS RT11SJ.SYS RT11FB.SYS RT11BL.SYS RT11XM.SYS
TT .SYS DT .SYS DP .SYS DX .SYS DY .SYS
RF .SYS RK .SYS DL .SYS DU .SYS DM .SYS
DS .SYS DD .SYS LP .SYS LS .SYS CR .SYS
CTRL/C CTRL/C
```

DIRECTORY  
/PRINTER

Volume directories can be printed on a line printer if one is available on your system. Check the answer to question 9 in the Hardware Configuration section of Chapter 2 to determine if your system has a line printer. Since listings print faster on a line printer than on the console terminal, it is to your advantage to use the line printer for large amounts of output. The /PRINTER option is used with the DIRECTORY command to cause a directory to be printed on the line printer instead of on the terminal. Make sure your line printer is turned on, and then type the DIRECTORY command as shown:

**Long and Short Command Format**

```
.DIRECTORY/PRINTER(RET)
```

The listing may be quite long. When the line printer has finished printing, retrieve the listing.

Initializing a storage volume clears its directory. A new (unused) volume should always be initialized before it is first used. In addition, any storage volume that contains files that are no longer needed can be initialized to recover the storage space. Note, however, that an initialize operation is used to remove all file names from the directory. So before you initialize a volume, make sure that it contains no files that you might want later.

Since you will use your storage volume to store several new files (created as a result of the various exercises in this manual), clear its directory using the monitor INITIALIZE command. This operation ensures that the volume has room for new files.

**Initializing the Storage Volume**

INITIALIZE

**Long Command Format**

```
.INITIALIZE(RET)  
Device? VOL:(RET)
```

(VOL: is the assigned logical device name for your storage volume.)

```
RK1:/Initialize; Are you sure?Y(RET)
```

**Short Command Format**

```
.INITIALIZE VOL:(RET)  
RK1:/Initialize; Are you sure?Y(RET)
```

The system prompt *physical-device-name/Initialize; Are you sure?* gives you an opportunity to verify the command. Typing a Y initiates the operation, while N stops the operation and returns control to the monitor command mode. Check your command line, make sure you are initializing your storage volume, and then type a Y. Again, list the directory of the storage volume. It should be empty.

**Long and Short Command Formats**

```
.DIRECTORY VOL:(RET)  
8-Jan-83  
  
0 Files, 0 Blocks  
4762 Free blocks
```

The number of blocks available for use on the volume is printed at the end of the directory and varies depending on the type of device you use as your storage volume.

The commands you have performed in this chapter have prepared the system for major operations that will follow. In Chapter 5 you begin by using the RT-11 editor to create text files that you will store on your initialized storage volume.

**SUMMARY:  
INITIAL  
MONITOR  
COMMANDS**

**ASSIGN** physical-device-name logical-device-name  
Assign a logical device name to a physical device name.

**DATE**  
Print the current date, if previously set.

**DATE** dd-mmm-yy  
Set the current date (day-month-year).

**DIRECTORY** ddn:  
List the volume directory on the terminal (ddn: is the mnemonic for the device name; the default storage volume, DK:, is assumed if ddn: is not specified).

**DIRECTORY/BRIEF** ddn:  
List a brief volume directory on the terminal, showing only file names.

**DIRECTORY/PRINTER** ddn:  
List the volume directory on the line printer.

**DIRECTORY/PRINTER/BRIEF** ddn:  
List a brief volume directory on the line printer.

**GT OFF**  
Disable the VT11 display hardware.

**GT ON**  
Enable the VT11 display hardware so that the graphics screen replaces the terminal printer/screen as the terminal output device.

**INITIALIZE** ddn:  
Clear the directory of the indicated volume (ddn: is the mnemonic for the device name and must be specified).

**SHOW**  
Print the status of all current logical device name assignments.

**TIME**  
Print the current time, if previously set.

**TIME** hh:mm:ss  
Set the current time-of-day (hour:minute:second).

**CTRL/C CTRL/C**

Interrupt the current operation or program and return control to monitor command mode.

**CTRL/E**

Direct terminal output to both the graphics screen and the terminal printer simultaneously. Type a second CTRL/E to return output control to the graphics screen only. (Valid only when VT11 display hardware is enabled.)

**CTRL/O**

Inhibit the remainder of output from printing on the terminal.

**CTRL/U**

Cancel every character in the current line.

**DELETE**

Cancel the last character typed on the current line.

**SUMMARY:  
SPECIAL  
CONTROL  
COMMANDS**

*LP11/LS11 Line Printer Manual* (EK-LP11-TM-005). Maynard, Mass.: Digital Equipment Corporation, 1975.

A hardware manual for the owners and operators of LP11/LS11 line printers and for those who will be programming computers to interact with these devices.

*RT-11 Mini-Reference Manual* (AA-M241A-TC). Maynard, Mass.: Digital Equipment Corporation, 1983.

A summary of all RT-11 monitor commands, command options, system utility program operating commands, and programmed requests.

*RT-11 System User's Guide* (AA-5279C-TC). Maynard, Mass.: Digital Equipment Corporation, 1983.

A guide to the use of the RT-11 operating system.

**REFERENCES**



## CHAPTER 5 CREATING AND EDITING TEXT FILES

The ability to create and edit text files is one of the most useful features of the RT-11 operating system. Not only can you create computer programs, data files, memos, and reports on line (that is, under the control of the system), but you can alter what you create without retyping the entire file.

You create and edit text files more often than you perform any other system operation. Therefore it is essential that you become familiar with the editing process as quickly as possible. Editing should become second nature to you as you learn to use the RT-11 computer system.

Two RT-11 editor system utility programs, EDIT.SAV and KED.SAV, are stored as part of the RT-11 operating system on your system volume. The demonstrations in this manual illustrate EDIT, which can be used on a video terminal or a hard-copy terminal. The use of KED is restricted to video terminals. For more information about KED, refer to the *PDP-11 Keypad Editor User's Guide*.

### THE RT-11 EDITOR

Text files that you create with the editor are stored in the computer in ASCII format. ASCII, which stands for the American Standard Code for Information Interchange, is an industry-standard code that consists of a numeric representation for each of the alphabetic characters (A to Z), the numeric characters (0 to 9), the punctuation characters, and some special communication control characters. When you type text on the terminal keyboard, the system automatically converts the text to ASCII codes; when you request listings on the terminal or line printer, the system converts the ASCII code back to the text characters.

The RT-11 editor uses a specially reserved area of computer memory to hold the text you are creating or editing. This area of memory is called the text buffer. When you create text, the characters that you type on the terminal keyboard are transmitted directly into the text buffer. When you edit text, the characters are copied from the input file into the text buffer, where you can modify them. When you have edited the text in the buffer to your satisfaction, the characters are moved out of the text buffer to the output file (Figure 5-1).

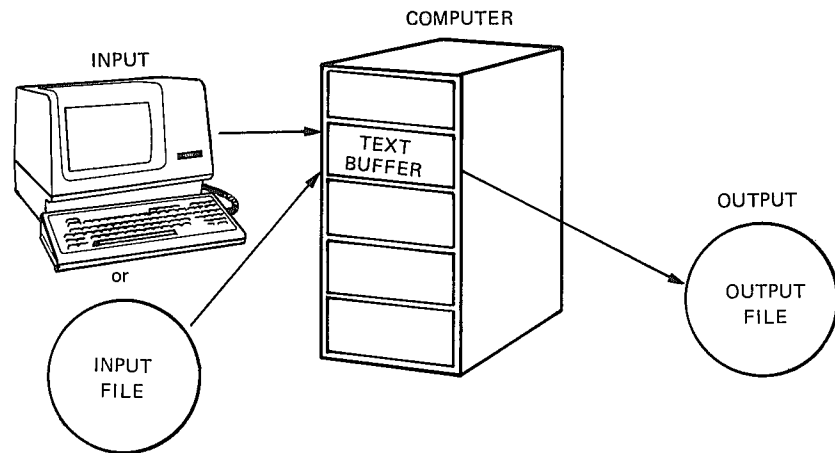


Figure 5-1 Editing with RT-11

Since the text buffer is a limited area of computer memory, you may at times try to input more text than the buffer can accommodate. If this condition becomes apparent to the editor, it prints a warning message on the terminal telling you that, before you can input any more text, you must make room in the buffer, either by transferring text to the output file or by erasing text already in the buffer.

You can avoid this inconvenience during editing if you make use of a concept called paging. When you create a large text file, instead of typing the file as one long stream of text, divide it into individual pages of approximately 50–60 lines in length; this corresponds roughly to the size of a line printer or terminal listing page. You can copy the text into and out of the buffer one page at a time. A single page of text is never too large for the text buffer and also fits on the line printer or terminal perforated paper when you obtain a listing.

## CREATING A TEXT FILE

EDIT/CREATE

You activate the editing capability by using the monitor `EDIT` command. When creating a file, you must use the `/CREATE` option followed by the file name and file type you want assigned to the new file. The default storage volume (`DK:`) serves as the default device, so unless you specify a device using one of the mnemonics in Table 4-2, the editor creates the new file on the device `DK:` (which is the system volume, unless changed via `ASSIGN`).

First, if you are using display hardware, disable it with the monitor `GT OFF` command; the editor has a special display capability that is not described until later in this chapter.

### Long and Short Command Format

```
.GT OFF(RET)
```

Next, use the editor to create a text file of five lines. Call the file DECIND.USA, and use the default storage volume — currently the same as the system volume — for the file.

### Long Command Format

```
,EDIT/CREATE(RET)
File? DECIND.USA(RET)
*
```

### Short Command Format

```
,EDIT/CREATE DECIND.USA(RET)
*
```

Once the output file is open (that is, when the appropriate file has been established for output operations), the editor prints a prompting asterisk at the left margin. The asterisk indicates that the editing command mode is in control. This prompt is your cue to enter an editing command.

The editing command used to create text is the I (Insert) command. Type:

```
*I
```

INSERT

All subsequent characters that you type on the terminal keyboard will now be entered into the text buffer just as you type them. Enter the following text exactly as shown, including all spaces and errors. Before you type the RETURN key, check the line to make sure that it matches what is shown here. Remember, if you make a typing mistake that is not intentional, you can use the DELETE key on the terminal keyboard to erase individual characters and the CTRL/U command to erase all characters on the current line. When you have finished typing the five lines, type the ESCAPE key twice. The ESCAPE key echoes on the terminal as a \$; it is used to execute an editing command and to return control to editing command mode.

ESCAPE ESCAPE

```
*IWE HOLD THESE TRUTS TO BEE SELF-EVIDENT,(RET)
  THAT ALL MEN ARE CREATED EQUAL, THAT THEY(RET)
  HAVE UNRELIABLE TENDENCIES OF WHICH THEY(RET)
  AR ENDOED BY THEIR CREATOR, THAT AMONG(RET)
  THESE ARE LIFE, LIBERTY AND HAPLENESS.(RET)
(ESC)(ESC)
*
```

Forget for the moment that this text contains several misspellings and other errors, and assume instead that you are satisfied with it and ready to transfer it from the text buffer to the output file. The EX (Exit) editing command performs this function. This command terminates editing, transfers all text in the text buffer to the output file, closes the currently open output

EXIT

file (making it unavailable for further output operations), and returns control to monitor command mode, indicated by a dot at the left margin. Use the EX command to close the file DECIND.USA:

```
*EX ESCESC
```

You now have a file on your system volume called DECIND.USA, consisting of the five lines of text you just created.

## EDITING A TEXT FILE

EDIT

The file DECIND.USA needs editing. To edit a file, you again use the EDIT command to activate the editor. Next indicate in the command line the two-letter device mnemonic for the volume on which the file resides (the default storage volume, DK:, is assumed). Following this, you indicate the file name and file type of the file. The editor then opens the file, making it available for input operations.

Thus, to open the file DECIND.USA for editing, type:

### Long Command Format

```
·EDIT RET  
File? DECIND.USA RET  
*
```

### Short Command Format

```
·EDIT DECIND.USA RET  
*
```

READ

The EDIT command opens the input (and output) files. Use the R (Read) editing command to read the first page of text from the input file into the text buffer. No output occurs to the output file, but the file is available for output at a later time. The input file itself is not altered in any way.

```
R ESCESC  
*
```

BEGINNING

Whenever text is read into the text buffer, a pointer is automatically positioned at the beginning of the text. This pointer is an invisible indicator that serves as a target for editing commands. The pointer pinpoints the exact location in the file where the next character will be inserted. For example, when you finished inserting text earlier (just before using the EX command), the pointer was positioned at the end of the file. Now that the EDIT command has been used to read text into the text buffer, the pointer is positioned at the beginning of the text in the text buffer. If the pointer is not at the beginning and you want to

move it there, you can use the B (Beginning) command; this command moves the pointer to the beginning of the text in the text buffer, no matter where the pointer is currently positioned:

```
B ESC ESC
*
```

With the pointer positioned at the beginning of the text buffer, you can use the L (List) editing command to list the text currently in the text buffer on your terminal printer. The List command lists text, starting at the pointer and continuing to whatever place you indicate by the command argument.

LIST

A command argument is simply a prefix to an editing command that sets limits on the command's actions. Command arguments are used frequently and are summarized in Table 5-1. Study this table for a moment before continuing.

**Table 5-1 Command Arguments**

Argument	Meaning
n	Represents any integer in the range -16383 to +16383; it may be preceded by a + or -. If no sign precedes n, it is assumed to be positive. Whenever an argument is acceptable in a command, its absence implies an argument of 1 (or -1 if only the - is present).
0	Refers to the beginning of the current line.
/	Refers to the end of text currently in the text buffer.

Thus, with the pointer positioned at the beginning of the text, use the / argument and the L command to list on the terminal all text in the buffer. The position of the pointer does not change. List the text and compare your output with the five lines shown in the following example — they should match exactly.

```
*/L ESC ESC
WE HOLD THESE TRUTHS TO BEE SELF-EVIDENT,
THAT ALL MEN ARE CREATED EQUAL, THAT THEY
HAVE UNRELIABLE TENDENCIES OF WHICH THEY
AR ENNDOWED BY THEIR CREATOR, THAT AMONG
THESE ARE LIFE, LIBERTY AND HAPLENESS.
*
```

If your output and the five lines above do not match exactly, then you probably typed some unintentional errors into DECIND.USA.

The remaining EDIT commands in this exercise depend upon an exact reproduction of DECIND.USA to function properly. Therefore, since you are not yet familiar with the EDIT commands necessary to correct your file, an existing copy of DECIND.USA with intentional errors must be substituted.

Prepare the text buffer by erasing it with CTRL/C(ESC)(ESC). This unusual command combination is required by the EDIT program when you want to exit without creating an output file. The structure of the command prevents you from accidentally eliminating a file with a single CTRL/C.

```
*CTRL/C(ESC)(ESC)
```

The monitor command mode period appears, signaling your departure from the editing command mode. Your system volume still contains the file DECIND.USA that you created earlier. However, it also contains the copy provided with the system, DEMOED.TXT, which you will use for the remainder of the exercise.

Before going any further, you must rename DEMOED.TXT to DECIND.USA to avoid confusion. A RENAME operation, explained fully in the File Copying Operations section of Chapter 7, is the method of choice. Type the following command:

```
,RENAME DEMOED.TXT DECIND.USA(RET)
```

The contents of DEMOED.TXT are now labeled DECIND.USA. Note, however, that if a file labeled DECIND.USA already exists and you rename another file to DECIND.USA, the system deletes the first file named DECIND.USA and renames the current one. Type EDIT DECIND.USA (RET) to open the file for input, and type the R command to read it into the text buffer.

```
,EDIT DECIND.USA(RET)  
*R(ESC)(ESC)
```

Since the pointer automatically returns to the beginning of the text with an R command, you can type /L to list the entire file.

```
*/L(ESC)(ESC)  
WE HOLD THESE TRUTHS TO BEE SELF-EVIDENT,  
THAT ALL MEN ARE CREATED EQUAL, THAT THEY  
HAVE UNRELIABLE TENDENCIES OF WHICH THEY  
AR ENDOWED BY THEIR CREATOR, THAT AMONG  
THESE ARE LIFE, LIBERTY AND HAPLENESS.  
*
```

JUMP

The text contains errors and misspellings deliberately introduced for the purposes of the exercises in this chapter. To correct the errors, reposition the pointer so that it is near the text you want to change. The J (Jump) command, for instance, in conjunction with a command argument, moves the pointer either backward or forward by the specified number of characters, including spaces. Type the J command now, using an argument of 18, to reposition the pointer 18 places ahead<sup>1</sup>:

```
*18J(ESC)(ESC)
*
```

Although you cannot see it, the pointer has moved from the beginning of the text buffer to the right of the 18th character. You can verify this by using the List command again. The List command with no argument prints from the pointer to the end of the current line and thus exposes the location of the pointer:

```
*L(ESC)(ESC)
S TO BEE SELF-EVIDENT,
*
```

The characters in the example should match the current line on your terminal, showing the pointer positioned at the first error in the text where an H is missing in the word TRUTS. Since the pointer is positioned between the second T and the S, use the Insert command to insert an H in the proper place:

```
*IH(ESC)(ESC)
*
```

Now use the V (Verify) command to verify the line. The V command, which does not require arguments, prints the entire line containing the pointer (the current line) on the terminal. It allows you to verify that a correction was properly made. The pointer is not moved as a result of the V command; its position remains just to the right of the last inserted character (shown here by the arrow):

```
*V(ESC)(ESC)
WE HOLD THESE TRUTHS TO BEE SELF-EVIDENT,
*                ↑
```

VERIFY

<sup>1</sup>Anytime you use the Jump command to move the pointer forward (or backward) by enough characters so that it moves to a new line, you must account for two extra characters in the command argument. This is because the editor treats the carriage return at the end of each line as two characters — a return and a line feed.

So far you have entered and executed editing commands one at a time. You can enter multiple commands by separating each individual command with a single ESCAPE. Typing two ESCAPEs then executes all the commands in the entire command string in consecutive order. For example, combine the J and L commands as shown in the following command string:

```
*7JESCLESCESC
E SELF-EVIDENT,
*
```

The 7J moves the pointer seven positions to the right, and L then lists the text from the pointer to the end of the line so that you can see the pointer's new position.

CTRL/X

A special CTRL command is available to erase multiple editing commands. The CTRL/X command (hold the CTRL key down and type the X key) causes the editor to ignore an entire command string that might extend over several lines if the I command is involved. The editor echoes with ^X, issues a carriage return, and prints an asterisk indicating that you are still in editing command mode and can enter a new command. For example, type:

```
*70JESCI START ARET
NEW LINE CTRL/X
*
```

In addition to the CTRL/X command, you may still use the DELETE key to erase individual characters in the command line one at a time, and the CTRL/U command to erase all characters entered on the current command line.

DELETE

Since you used the CTRL/X to ignore this last command string, the pointer is still positioned at the next error in the file — just before the extra E in the word BEE. You can erase this extra character by using the D (Delete) command.<sup>1</sup> The D command removes one character (or space) to the right of the pointer for every +1 in its argument and one character to the left for every -1. Use the D command to erase the extra E and then verify the line (+1 is assumed if no argument is used):

```
*DESCESCESC
WE HOLD THESE TRUTHS TO BE SELF-EVIDENT,
*
↑
```

<sup>1</sup>The Delete command should not be confused with the DELETE key on the terminal keyboard. While both perform the delete function, the D command is used to erase characters already within a text file; the DELETE key is used to erase typed characters in a command string or during text creation.

As you can see from the position of the pointer in the example (shown by the arrow), the D command does not actually move the pointer, but simply erases characters around the pointer. Since the extra E was erased, the pointer is now positioned between the E and the space.

Just as you can use the Jump command to move the pointer by characters, you can use the A (Advance) command to move the pointer by entire lines. Again you give the command an argument that indicates the number of lines, either forward or backward. The pointer is positioned at the beginning of the new line. Use the A command to move the pointer forward two lines, and then list the current line:

ADVANCE

```
*2A(ESC)L(ESC)(ESC)
HAVE UNRELIABLE TENDENCIES OF WHICH THEY
*
```

This entire line does not belong in the text. To erase it, you could count the number of characters in the line and use this number as an argument to the D command; however, there is an easier way. The K (Kill) command erases the entire line following the pointer and positions the pointer at the beginning of the next line in the text. Type:

KILL

```
*K(ESC)L(ESC)(ESC)
AR ENDOWED BY THEIR CREATOR, THAT AMONG
*
```

The pointer is now at the beginning of the next line in the text. As you can see, this line also contains an error, the word ARE is incorrectly spelled as AR. Use the J command to jump over two characters, and insert the E. Then verify the line:

```
*2J(ESC)IE(ESC)V(ESC)(ESC)
ARE ENDOWED BY THEIR CREATOR, THAT AMONG
* ↑
```

The arrow shows where the pointer is now positioned. This line still contains an error — it is missing the words WITH CERTAIN INALIENABLE RIGHTS, which should follow the word CREATOR. You can count the number of characters from the pointer to the second R in CREATOR and then jump the pointer by this number, or you can use the G (Get) command. The G command searches, from the pointer, for the first occurrence of a specified character string and leaves the pointer at the end of that string. Use the G command to search for the string OR (in CREATOR); then insert the missing words and list the lines that have changed. Notice how you use the carriage return to

GET

break the line into two parts (the `SP` symbol is used to show where you should insert spaces):

```
*GORESCISPWITHSPCERTAINRET
INALIENABLESPRIGHTSESC-AESC2LESCESC
ARE ENDOWED BY THEIR CREATOR WITH CERTAIN
INALIENABLE RIGHTS, THAT AMONG
*
```

To list both lines, it was necessary to move the pointer back to the beginning of the first line you changed; this was done by the `-A` command. The `2L` command then listed both lines. Notice where the pointer is; it was moved by the `-A` command and was not repositioned by the `L` command.

You must be careful when you use the Get command, because the character string you specify must be unique if you want the pointer to move to the correct spot. For example, if the characters `OR` had occurred anywhere after the pointer and before the word `CREATOR`, the pointer would have stopped there instead, and you would have inserted text in the wrong place.

The final errors in this text occur in the last line. The words `THE PURSUIT OF` are missing, and the word `HAPLENESS` is a misspelling. Use the Get command to move the pointer to the word `AND` and insert the missing text. Move the pointer again with the Get command to the `PLE` of `HAPLENESS`; erase the `LE`, and insert `PI`. Then verify the line:

```
*GANDESCISPTHESPPURSUITSPOFESCESC
*GPLEESC-2DESCIPESCIVESCESC
THESE ARE LIFE, LIBERTY AND THE PURSUIT OF HAPPINESS.
*
```

CTRL/L

Large text files — 50 lines or more — should be delimited into pages. To do this, insert a form feed into the text at the place where you want the page to end. A form feed is typed as a `CTRL/L` (hold the `CTRL` key down and type the `L` key), which the editor recognizes as a page break.

Since this text file is only five lines long, there is really no need to delimit it as a page. However, for the sake of practice, insert a form feed at the end of this file. Then move the pointer to the beginning of the text buffer and list the entire text. Compare your text with the following example. If errors remain in your file, fix them by using the commands described so far.

```
*G.ESCIRET
CTRL/L (CTRL/L echoes as eight line feeds.)
```

```

ESC B ESC /L ESC ESC
WE HOLD THESE TRUTHS TO BE SELF-EVIDENT,
THAT ALL MEN ARE CREATED EQUAL, THAT THEY
ARE ENDOWED BY THEIR CREATOR WITH CERTAIN
INALIENABLE RIGHTS, THAT AMONG
THESE ARE LIFE, LIBERTY AND THE PURSUIT OF HAPPINESS.

```

\*

This text is correct in spelling and content, but the last two lines should be justified to make them easier to read. The pointer is currently at the beginning of the text. Use the G command to search for the character string AMONG; then insert and delete text to justify the lines. Finally, list the text again:

```

*GAMONGESC I SP THESE SP ARE ESC A ESC 10D ESC B ESC /L ESC ESC
WE HOLD THESE TRUTHS TO BE SELF-EVIDENT,
THAT ALL MEN ARE CREATED EQUAL, THAT THEY
ARE ENDOWED BY THEIR CREATOR WITH CERTAIN
INALIENABLE RIGHTS, THAT AMONG THESE ARE
LIFE, LIBERTY AND THE PURSUIT OF HAPPINESS.

```

\*

Once you are satisfied with your text, you are ready to transfer it to the output file. You could use the EX command to transfer the text, as you did earlier in the section Creating a Text File. However, suppose your input file has additional pages of text that require editing. If you use the EX command, all remaining text in the input file will be read through the text buffer into the output file, and the files will be closed although you may want to do more editing. To avoid this, you can use the N (Next) command. This command transfers the text currently in the text buffer to the output file, clears the text buffer, and reads in the next page from the input file. The pointer is positioned at the beginning of the text buffer.

NEXT
------

```

*N ESC ESC
?EDIT-F-End of input file
*          (No text remains in the input file.)

```

If you use the N command when no text remains in the input file (as just happened), the editor prints a message on the terminal telling you so. At this point, you can type the EX command to close the file.

```

*EX ESC ESC

```

When you close a file after editing, the editor creates a file on the default storage volume (or system volume). It gives this new file the file name and file type that you indicated for input. It then renames the input file so that the file retains its file name but is assigned a file type of .BAK. This file type identifies a backup file, here an original input file retained in case of editing mistakes or accidental deletion of the new file. Thus you now have two versions of the DECIND file on your system volume: DECIND.USA, which is the edited version, and DECIND.BAK, which is the unedited (original) input file. Verify this by using the monitor DIRECTORY command:

### Long and Short Command Format

```
.DIRECTORY DECIND.*(RET)
 08-Jan-83
DECIND.BAK  1 08-Jan-83  DECIND.USA  1 08-Jan-83
 2 Files, 2 Blocks
 496 Free blocks
```

The asterisk (\*) following DECIND. is a type of shorthand notation called wildcard construction. Here it means to list all files named DECIND, regardless of their file type. Wildcard construction is explained in detail in the Multiple File Operations section of Chapter 7.

Whenever you edit the same file a number of times, new versions overwrite old versions. Thus only two versions of the edited file (filnam.BAK and filnam.typ) ever reside on a volume at one time.

## USING UPPERCASE AND LOWERCASE CHARACTERS

Later model terminals (for example, LA120 DECwriters and VT100 video terminals) have the capability to print in uppercase and lowercase. Certain line printers also have this capability. You can use the uppercase/lowercase capability of these devices if you type the EL (Edit Lower) editing command before entering the text you want to insert in lowercase. The EL command instructs the system to accept all characters typed as they appear on the keyboard. The monitor facility, which converts all alphabetic characters to uppercase, is disabled. In addition, the characters are echoed on the terminal printer or screen as uppercase and lowercase characters.

Open the file DECIND.USA again, and type the EL command:

### Long and Short Command Format

```
.EDIT DECIND.USA(RET)
*EL (ESC) (ESC)
*
```

Edit Lower

Once you have typed the EL command, you can use the SHIFT key on the terminal to designate uppercase, just as you do on a typewriter. Editing commands may be entered as either uppercase or lowercase characters. For example, type the following commands, which change the characters in the first line of the file DECIND.USA to uppercase and lowercase:

```
*r(ESC)b(ESC)l(ESC)(ESC)
WE HOLD THESE TRUTHS TO BE SELF-EVIDENT,
*k(ESC)iWe hold these truths to be self-evident,(RET)
(ESC)-a(ESC)v(ESC)(ESC)
We hold these truths to be self-evident,
*
```

The uppercase and lowercase capability is useful for reports, memos, and other textual material that you list on uppercase/lowercase devices. However, all characters are printed as uppercase if you list the file on a line printer or terminal that does not have the uppercase/lowercase capability.

Edit Upper

If at any time you want to revert to strictly uppercase editing, type the EU (Edit Upper) command:

```
*e u(ESC)(ESC)
*
```

Uppercase editing is a default mode. Whenever you open a file for editing or create a new file, you must enter the EL command if you want to use the uppercase/lowercase capability.

Close the file DECIND.USA by typing:

```
*E X(ESC)(ESC)
```

#### EDIT filespec

Activate the editor and open the file for editing.

#### EDIT/CREATE filespec

Activate the editor and create a new file.

#### Control Commands

##### CTRL/L

Insert a form feed. The form feed character is used to delimit pages of text in a file (introduced as part of text by the Insert command).

##### CTRL/X

Ignore all commands in the current editing command string.

**SUMMARY:  
EDITING  
COMMANDS**

## Command Arguments

n(+ or -)

An integer value between -16383 and +16383 that sets the range of a command's actions based on the pointer's current position.

0

Beginning of the current line (the line containing the pointer).

/

End of the text in the text buffer.

## Input/Output Commands (pointer is not repositioned) (x indicates that an argument can be used)

EX

Exit; terminate editing, transfer the contents of the text buffer and the remainder of input file to the output file; close input and output files; return to monitor command mode.

xL

List; list, from the pointer, x lines of text.

xN

Next; write the contents of the text buffer to the output file, clear the text buffer, and read into it the next page from the input file; perform this write/read sequence x times.

V

Verify; list the current line (the line containing the pointer) on the terminal.

## Pointer Location Commands (pointer is repositioned) (x indicates that an argument can be used)

xA

Advance; move the pointer to the beginning of the xth line from the current pointer position.

B

Beginning; move the pointer to the beginning of the text buffer.

xJ

Jump; move the pointer forward or backward by x characters.

**Text Modification Commands** (pointer is repositioned)  
(x indicates that an argument can be used)

**xD**

Delete; erase x characters to the right (or left) of the pointer.

**I text** **ESC**

Insert; insert text into the text buffer at the present pointer position.

**xK**

Kill; erase x lines of text, beginning at the pointer.

**Search Command** (pointer is repositioned)

(x indicates that an argument can be used)

**xG text**

Get; search the text buffer, beginning at the pointer, for the xth occurrence of the indicated text string and leave the pointer at the end of the text string.

**Uppercase/Lowercase Commands** (pointer is not affected)

**EL**

Edit Lower; accept characters typed at the keyboard as uppercase/lowercase.

**EU**

Edit Upper; revert to uppercase editing (after EL).

If your system configuration includes VT11 display hardware, there are several advantages to your using it during editing.<sup>1</sup> First, the graphics screen becomes a window into the text buffer, exposing twenty lines of text at a time: the current line, the ten lines preceding it, and the nine lines following it. Figure 5-2 illustrates this format. As you edit, the lines in view shift to conform to the current line. In addition, the pointer is visible and appears as a blinking, L-shaped cursor. Its position is automatically adjusted as you execute editing commands. Finally, the four lines at the bottom of the screen display the last three command lines plus the current command line. Horizontal dashes separate the text of the file from your commands.

---

<sup>1</sup>If your system does not have VT11 display hardware, skip to the next section, entitled Creating the Demonstration Programs.

**USING A  
GRAPHICS DISPLAY  
TERMINAL  
DURING EDITING**

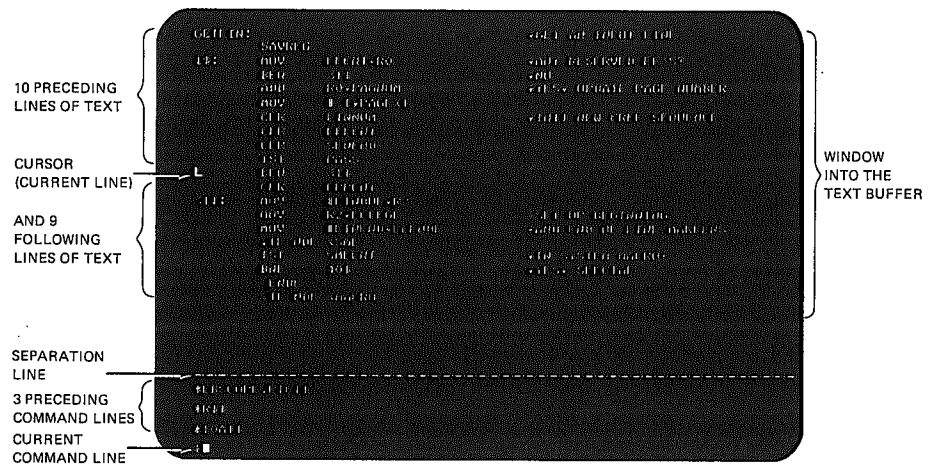


Figure 5-2 Text Window Format

### Normal Use of the Graphics Display

All editing commands and functions described so far can be used when the graphics screen is enabled. The only difference is that terminal I/O is rearranged on the screen as shown in Figure 5-2. Note that the L and V editing commands become superfluous since the pointer is always displayed on the screen. Also, since twenty lines of text are always displayed, any List command within that range is unnecessary.

Currently, your graphics screen is not enabled. To enable it, use the monitor GT ON command as you did in Chapter 4:

### Long and Short Command Format

```
.GT ON(RET)
```

Now when you use the EDIT command to activate the editor, the graphics screen will be rearranged as shown in Figure 5-2. You can use the CTRL/E command, described in Chapter 4, to request simultaneous I/O on the terminal printer and graphics screen.

### Immediate Mode

In addition to the regular editing capability, a quick and easy method of graphics editing, called immediate mode, is available. Immediate mode uses a simplified set of editing commands that are limited to pointer relocation and character deletion and insertion. Most of these commands are similar to the special CTRL commands because to type them you use the CTRL key in combination with another character key. How-

ever, the use of these particular control commands is meaningful only in the editor immediate mode. Table 5-2 lists the commands.

**Table 5-2 Immediate Mode Commands**

Command	Meaning
CTRL/N	Advance the cursor to beginning of next line (equivalent to A).
CTRL/G	Move the cursor to the beginning of the previous line (equivalent to -A).
CTRL/D	Move the cursor forward by one character (equivalent to J).
CTRL/V	Move the cursor back by one character (equivalent to -J).
DELETE	Delete the character immediately preceding the cursor (equivalent to -D).
ESCAPE	Return control to the editing command mode.
double ESCAPE	Summon immediate mode.

Use the editor to open a new file called IMMODE.TXT:

**Long and Short Command Format**

```
.EDIT/CREATE IMMODE.TXTRET
*
```

Now activate immediate mode. You do this by typing the ESCAPE key twice in response to the editing command mode asterisk. Since there are no other commands in the command line, the editor recognizes the double ESCAPE as an immediate mode command.

ESCAPE ESCAPE

```
*ESCESC
|
```

The editor responds by printing an exclamation point in the command portion of the screen; the exclamation point signifies that you are using immediate mode.

Character insertion is the default operation and occurs whenever you type a character other than one of the immediate mode commands listed in Table 5-2.

Character Insertion



CTRL/D, equivalent to J, moves the cursor forward one character; type CTRL/D ten times:

CTRL/D

CTRL/D

(ten [10] times)

THE SLINGS OF OUTRAGEDOUS FORTUNE,

Next type this text (it will be inserted immediately to the left of the cursor):

SPANDSPARROWS

The text on the screen should now look as follows:

TO BE OR NOT TO BE - THAT IS THE QUESTION;  
WHETHER 'TIS NOBLER IN THE MIND TO SUFFER  
THE SLINGS AND ARROWS OF OUTRAGEDOUS FORTUNE,  
OR TAKE ARMS AGAINST A SEA OF TROUBLES,  
AND BY OPPOSING END THEM?

Check your results and correct any other mistakes you may notice.

To return to the standard editing command mode, type a single ESCAPE.

ESCAPE

ESC

\*

This ESCAPE command does not echo on the screen. Notice that the exclamation point immediately disappears and the text window format returns; an asterisk appears immediately below the exclamation point on the screen.

You use immediate mode only to create and edit text. Operations that move text in and out of the text buffer must be done with standard editing commands.

You do not need to save the text you have just created, so use the CTRL/C command and two ESCAPEs to return control directly to monitor command mode. As mentioned before, EDIT requires this unusual command combination to prevent an accidental CTRL/C from killing your text.

CTRL/C ESCAPE  
ESCAPE

CTRL/C ESC ESC

Following are two demonstration programs. One is written in the FORTRAN IV programming language and one is written in the MACRO-11 assembly language. Both programs are used in later chapters of this manual, and both contain intentional misspellings and errors.

**CREATING THE  
DEMONSTRATION  
PROGRAMS**

Use the editor to create these programs. Type them exactly as they are shown, including errors. Use tabs and spaces to format each line as shown (remember that tab stops are positioned every eight spaces across the terminal page). Make sure that the FORTRAN IV program is formatted properly so that a source comparison described in the next chapter will operate properly. Except for the comment lines (those beginning with a C) and the lines that begin with a number, begin all lines with a tab. Use any of the editing commands described in this chapter. Activate the display editor and immediate mode if you wish.

When you have finished, check each file carefully. The two files should match those shown here exactly, including tabs and spaces. Correct any errors that you find that are not intentional. Obtain a listing of each file by using B `ESC/L ESCESC` before closing the file.

Create the FORTRAN IV file first. Call it GRAPH.FOR and use the system volume for storage. Then create the MACRO-11 program. Call it SUM.MAC and again use the system volume for storage.

#### NOTE

Knowledge of the FORTRAN IV and MACRO-11 languages is not necessary to create these demonstration programs.

The following program, GRAPH.FOR, is the FORTRAN IV demonstration program.

```

C GRAPH.FOR      VERSION 1
C THIS PROGRAM PRODUCES A PLOT ON THE TERMINAL
C OF AN EXTERNAL FUNCTION, FUN(X,Y)
C THE LIMITS OF THE PLOT ARE DETERMINED BY THE DATA STATEMENTS
C "STAB" IS FILLED WITH A TABLE OF WEIGHT FLAGS
C "STRING" IS USED TO BUILD A LINE OF GRAPH FOR PRINTING

      SCAL(ZMIN,ZMAX,MAXZ,K)=ZMIN+FLOAT(K-1)*(ZMAX-ZMIN)/FLOAT(MAXZ-1)
      LOGICAL*1 STRING(13,3),STAB(100)
      DATA XMIN,XMAX,MAXX/-5,5,45/
      DATA YMIN,YMAX,MAXY/-5,5,72/
      DATA FMIN,FMAX/0.0,1.0/
      CALL SCOPY(' - 1 2 3 4 5 6 7 8 9 +',STAB)
      MAXFLEN(STAB)
      DO 20 IX=1,MAXX
        X=SCAL(XMIN,XMAX,MAXX,IX)
        CALL REPEAT('*',STRING,MAXY)
        IF(IX,EQ,1 .OR. IX,EQ,MAXX) GOTO 20
        DO 10 IY=2,MAXY-1
          Y=SCAL(YMIN,YMAX,MAXY,IY)
          IFUN=2+INT(FLOAT(MAXF-3)*(FUN(X,Y)-FMIN)/(FMAX-FMIN))
10          STRING(IY)=STAB(MINO(MAXF,MAXO(1,IFUN)))
30          CALL PUTSTRING(7,STRING,' ')
      CALL EXIT
      END

      FUNCTION FUN(X,Y)
      R=SQRT(X**2+Y**2)
      FUN=X*Y*R*EXP(-R)**2
      RETURN
      END

```

The following program, SUM.MAC, is the MACRO-11 demonstration program.

```

.TITLE SUM.MAC VERSION 1

.MCALL .TTYOUT, .EXIT, .PRINT

      N = 70,          ;NO. OF DIGITS OF 'E' TO CALCULATE

;      'E' = THE SUM OF THE RECIPROCAL OF THE FACTORIALS
;      1/0! + 1/1! + 1/2! + 1/3! + 1/4! + 1/5! + ...

EXP:  .PRINT #MESSAG      ;PRINT INTRODUCTORY TEXT
      MOV #N,R5           ;NO. OF CHARS OF 'E' TO PRINT
FIRST: MOV #N+1,R0        ;NO. OF DIGITS OF ACCURACY
      MOV #A,R1           ;ADDRESS OF DIGIT VECTOR
SECOND: ASL @R1            ;DO MULTIPLY BY 10 (DECIMAL)
      MOV @R1,-(SP)        ;SAVE *2
      ASL @R1              ;*4
      ASL @R1              ;*8
      ADD (SP)+,(R1)+      ;NOW *10, POINT TO NEXT DIGIT
      DEC R0               ;AT END OF DIGITS?
      BNE 2ND              ;BRANCH IF NOT
      MOV #N,R0            ;GO THRU ALL PLACES, DIVIDING
THIRD: MOV -(R1),R3        ;BY THE PLACES INDEX
      MOV #-1,R2           ;INIT QUOTIENT REGISTER
FOURTH: INC R2             ;BUMP QUOTIENT
      SUB R0,R3            ;SUBTRACT LOOP ISN'T BAD
      BCC FOURTH           ;NUMERATOR IS ALWAYS < 10*N
      ADD R0,R3            ;FIX REMAINDER
      MOV R3,@R1           ;SAVE REMAINDER AS BASIS
                          ;FOR NEXT DIGIT
      ADD R2-2(R1)         ;GREATER INTEGER CARRIES
                          ;TO GIVE DIGIT
      DEC R0               ;AT END OF DIGIT VECTOR?
      BNE THIRD           ;BRANCH IF NOT
      MOV -(R1),R0         ;GET DIGIT TO OUTPUT
FIFTH: SUB #10.,R0         ;FIX THE 2.7 TO .7 SO
                          ;THAT IT IS ONLY 1 DIGIT
                          ;(REALLY DIVIDE BY 10)
      BCC FIFTH           ;MAKE DIGIT ASC II
      ADD #10+'0',R0       ;OUTPUT THE DIGIT
      .TTYON
      CLR @R1              ;CLEAR NEXT DIGIT LOCATION
      DEC R5               ;MORE DIGITS TO PRINT?
      BNE FIRST           ;BRANCH IF YES
      .EXIT                ;WE ARE DONE

EXP:  .REPT N+1
      .WORD 1              ;INIT VECTOR TO ALL ONES
      .ENDR

MESSAG: .ASCII /THE VALUE OF E IS:/ <15><12> /2./ <200>
      .EVEN

      .ENDEXP

```

When you have created and checked these two programs, obtained listings, and stored them as files on your system volume, go on to Chapter 6, *Comparing Text Files*. Chapter 6 demonstrates a proofreading aid that helps you evaluate your editing ability.

*RT-11 System User's Guide (AA-5279C-TC)*. Maynard, Mass.: Digital Equipment Corporation, 1983.

**REFERENCE**

A guide to the use of the RT-11 operating system.



## CHAPTER 6 COMPARING TEXT FILES

The RT-11 operating system provides a proofreading aid, called a source comparison, to help you quickly establish the differences between two ASCII text files. During a source comparison, the system compares the two files, character for character, and prints on the terminal (or line printer) any lines that contain differences.

Usually, you perform a source comparison against two files that you expect to be the same, or at least similar. For example, if an individual has copied one of your files to make changes to it, you can quickly scan the changes by performing a source comparison between the new version and your original. Another use of a source comparison is to check edits you have made to a file yourself. By comparing the backup file against the edited version, you can proofread the changes since only the portions of text that are different are printed.

In this chapter, you will use source comparisons to find editing errors that may exist in the demonstration programs (GRAPH.FOR and SUM.MAC) that you created in Chapter 5. These demonstration programs contain intentional misspellings and misplaced text that you must correct before the programs can be used in later demonstrations. On your system volume is a counterpart of each file. These counterparts are provided as part of the RT-11 operating system so that you can use them to perform a source comparison against your own versions. Essentially, the counterpart programs have been carried one step further in the editing process than your own; they contain no editing errors. Therefore, when you compare them against your versions, the printed list of differences will reflect the typing errors that still exist in your versions — some of these errors are intentional; others you may have inadvertently introduced during editing. All must be corrected before you can use the programs.

The monitor command used to compare two text files is the DIFFERENCES command. When you type this command on the terminal, it activates the RT-11 utility program called SRCCOM.SAV, which is part of the RT-11 operating system stored on the system volume. The system prompts you for the input file names. Respond to the input prompts with the names of the files you want to compare; the default storage volume is the system volume. The output will be sent to the terminal, which is the default device for output.

### PERFORMING A COMPARISON

## DIFFERENCES

The programs that you created in Chapter 5, SUM.MAC and GRAPH.FOR, have their respective counterparts, DEMOX1.MAC and DEMOF1.FOR, on the system volume. Use the DIFFERENCES command to compare the MACRO-11 (.MAC) files first. The /MATCH option indicates the number of lines that determine a “match”, explained in a moment.<sup>1</sup>

## Long Command Format

```
.DIFFERENCES/MATCH:1(RET)
File 1? DEMOX1.MAC(RET)
File 2? SUM.MAC(RET)
```

## Short Command Format

```
.DIFFERENCES/MATCH:1 DEMOX1.MAC SUM.MAC(RET)
```

The list of differences printed on your console terminal should be similar to the following example. It will show all the differences listed here, plus any others that you may have introduced yourself during editing.

Notice the format of the list. Individual sections are marked with the letters A, B, C, and D to help you become acquainted with the format. A description follows the list, and you should refer to it as you study the list.

```
A 1) DK:DEMOX1.MAC
A 2) DK:SUM.MAC
*****
C 1)1      .TITLE EXAMP.MAC      (VERSION PROVIDED)
1)
D 1)      .MCALL .TTYOUT, .EXIT, .PRINT
B ****
C 2)1      .TITLE SUM.MAC VERSION 1
2)
D 2)      .MCALL .TTYOUT, .EXIT, .PRINT
*****
C 1)1      BNE      SECOND      ;BRANCH IF NOT
D 1)      MOV      #N,RO        ;GO THRU ALL PLACES,
;DIVIDING
B ****
C 2)1      BNE      2ND         ;BRANCH IF NOT
D 2)      MOV      #N,RO        ;GO THRU ALL PLACES,
;DIVIDING
*****
C 1)1      ADD      #10+'0,RO    ;MAKE DIGIT ASCII
D 1)      .TTYON                ;OUTPUT THE DIGIT
B ****
C 2)1      ADD      #10+'0,RO    ;MAKE DIGIT ASC II
D 2)      .TTYON                ;OUTPUT THE DIGIT
*****
C 1)1      .END      EXP
B ****
C 2)1      .ENDEXP
*****
?SRCCOM-W-Files are different
```

<sup>1</sup>Users of display hardware may wish to enable both the graphics screen and the terminal printer by first typing the CTRL/E command.

The first two lines identify the two files that are being compared. The file name and the device on which the file resides are printed. Also, the numbers 1) and 2) are assigned to the files (see lines labeled A in the example list above).

The first difference that is listed occurs in the title line of the program. Usually differences that occur in these two lines are intentional and reflect information that is unique to each file, such as name and file type, version or edit number, and perhaps date of creation.

The numbers that appear at the left margin of the list further identify the files. For example, 1)1 indicates the first page of the first file and 2)1 indicates the first page of the second file.

The lines of both files are compared character for character. Blank lines are ignored, but all other characters, including tabs and spaces, are compared. When two lines are found to be different, the system prepares a difference section, which it subsequently prints (see lines labeled B).

The system prepares the difference section as follows. When it finds two lines that are different, it notes the page number and records the lines (see lines labeled C). Next it searches for a match. A match is a certain number of lines in each file that are exactly the same. Since you specified a match of 1 in the /MATCH:n option (/MATCH:1), the system in this case searches for a single line in each file that is exactly the same. When the system finds a match, it records the last line of the match for identification purposes (see lines labeled D). Then it prints the difference section and repeats the process, preparing a subsequent difference section if more differences exist. Individual difference sections are separated from each other by a long row of asterisks, while the short rows of asterisks separate the lines of the first file from those of the second.

DIFFERENCES/  
MATCH:n

A message is printed following the comparison. *Files are different* is printed if differences exist; *No differences found* is printed if the files are the same.

Check the list printed on your terminal to find the errors the system detected. Mark each error on the listing of SUM.MAC that you obtained in Chapter 5.

Now perform a source comparison between the FORTRAN IV files, DEMOF1.FOR and GRAPH.FOR.

### Long Command Format

```
DIFFERENCES/MATCH:1 (RET)
File 1? DEMOF1.FOR (RET)
File 2? GRAPH.FOR (RET)
```

## Short Command Format

```
.DIFFERENCES/MATCH:1 DEMOF1.FOR GRAPH.FOR␣
1) DK:DEMOF1.FOR
2) DK:GRAPH.FOR
*****
1)1 C EXAMP.FOR      (VERSION PROVIDED)
1) C THIS PROGRAM PRODUCES A PLOT ON THE TERMINAL
****
2)1 C GRAPH.FOR     VERSION 1
2) C THIS PROGRAM PRODUCES A PLOT ON THE TERMINAL
*****
1)1 C "STAB" IS FILLED WITH A TABLE OF HEIGHT FLAGS
1) C "STRING" IS USED TO BUILD A LINE OF GRAPH FOR PRINTING
****
2)1 C "STAB" IS FILLED WITH A TABLE OF WEIGHT FLAGS
2) C "STRING" IS USED TO BUILD A LINE OF GRAPH FOR PRINTING
*****
1)1                MAXF=LEN(STAB)
1)                DO 20 IX=1,MAXX
****
2)1                MAXFLEN(STAB)
2)                DO 20 IX=1,MAXX
*****
1)1    30          CALL PUTSTR(7,STRING,' ')
1)          CALL EXIT
****
2)1    30          CALL PUTSTRING(7,STRING,' ')
2)          CALL EXIT
*****
?SRCCOM-W-Files are different
```

Similarly, mark the errors on the listing of GRAPH.FOR that you obtained in Chapter 5.

Now return to the section entitled Editing a Text File in Chapter 5. Review the editing commands described there and the summary at the end of the section. Use the appropriate commands to correct the files SUM.MAC and GRAPH.FOR. When you have finished editing, perform the source comparisons again against DEMOX1.MAC and DEMOF1.FOR. If you have edited the files correctly, the comparison finds differences only between the first lines of each program. The following messages should print on your console:

```
.DIFFERENCES/MATCH:1␣
File 1? DEMOF1.FOR␣
File 2? GRAPH.FOR␣
1) DK:DEMOF1.FOR
2) DK:GRAPH.FOR
*****
1)1    C EXAMP.FOR      (VERSION PROVIDED)
1)    C THIS PROGRAM PRODUCES A PLOT ON THE TERMINAL
****
2)1    C GRAPH.FOR     VERSION 1
2)    C THIS PROGRAM PRODUCES A PLOT ON THE TERMINAL
*****
?SRCCOM-W-Files are different
```

and

```
.DIFFERENCES/MATCH:1(RET)
File 1? DEMOX1.MAC(RET)
File 2? SUM.MAC(RET)
1) DK:DEMOX1.MAC
2) DK:SUM.MAC
*****
1)1                .TITLE EXAMP.MAC (VERSION PROVIDED)
1)
1)                .MCALL .TTYOUT, .EXIT, .PRINT
****
2)1                .TITLE SUM.MAC VERSION 1
2)
2)                .MCALL .TTYOUT, .EXIT, .PRINT
*****
?SRCCOM-W-Files are different
```

These messages indicate that a difference exists in the first line of each program. However, no other differences were found in the programs during the comparison. Thus, your programs are ready for use in later demonstrations, and you know how to create and edit programs.

If differences still exist in your files and you cannot resolve them by editing, you may continue to the next chapter if you wish. However, you need practice editing, and it is to your advantage to rework the examples in both Chapter 5 and this chapter.

## DIFFERENCES

List the differences between two ASCII text files.

### DIFFERENCES/MATCH:n

Indicate the number of lines (n) to determine a match; the default number is 3.

**SUMMARY:  
COMPARISON  
COMMAND**

*RT-11 System User's Guide* (AA-5279C-TC). Maynard, Mass.: Digital Equipment Corporation, 1983.

A guide to the use of the RT-11 operating system.

*RT-11 System Utilities Manual* (AA-M239A-TC). Maynard, Mass.: Digital Equipment Corporation, 1983.

A guide to the use of the RT-11 system utilities.

**REFERENCES**

)

)

)

)

)

## CHAPTER 7

# PERFORMING FILE MAINTENANCE OPERATIONS

The system volume, as it is initially supplied, contains only the files of the RT-11 operating system — the monitor files, the system device handlers, the system utility programs, and perhaps the language processors. Since the system volume serves as the default storage volume for all system operations (unless DK: was assigned to another volume), you will discover that it acquires many additional files during normal use. For example, files that you create with the editor are written on the system volume; edited files automatically create backup versions on the system volume; many utility programs create output and listing files on the system volume as part of their normal processing operations. By the time you finish an average session of computer operations, several new file names have been added to the directory of your system volume. Eventually your system volume may become full and its directory cluttered with the names of files for which you have no use. To avoid this you should perform regular housekeeping, or file maintenance, operations as you use the system. You should update and transfer copies of your important files to other storage volumes for safekeeping and later use, and you should delete from your system and storage volume directories the names of files you no longer need.

The RT-11 operating system provides a number of monitor commands for this purpose. These commands activate the RT-11 utility programs called PIP.SAV, DUP.SAV, and DIR.SAV, which are part of the RT-11 operating system stored on your system volume. These utility programs allow you to transfer and erase files. The commands used in this chapter show one way to maintain your system and storage volume. When you become more familiar with system operations and learn some of the commands not described here, you may prefer other methods.

Before you perform operations that might move or erase files on a volume, list a directory of the volume involved. The directory tells you the full names of files, their sizes, and whether backup copies exist. A directory of your system volume shows the files that have been added to it through normal use.

First obtain a directory of your system volume (as you learned in Chapter 4), using the appropriate command to list it on either the terminal or the line printer. The directory is relatively long; let it list to completion.

### FILE DIRECTORY OPERATIONS

## Long and Short Command Format

(Line printer)

```
.DIRECTORY/PRINTERRET
```

(Terminal)

```
.DIRECTORYRET
```

At the end of the system volume directory you should see several additional entries. These files are the result of the system operations you have performed so far:

DECIND,USA	1	8-JAN-83
DECIND,BAK	1	8-JAN-83
GRAPH .FOR	2	8-JAN-83
GRAPH .BAK	2	8-JAN-83
SUM .MAC	3	8-JAN-83
SUM .BAK	3	8-JAN-83

Next list a brief directory of your storage volume. This directory should be empty (void of any file names or file types) because you initialized it in Chapter 4.

## Long and Short Command Format

(Line printer)

```
.DIRECTORY/BRIEF/PRINTER VOL:RET
```

(Terminal)

```
.DIRECTORY/BRIEF VOL:RET
```

These directories give you the information you need for erasing and copying files. For example, you know the additional files that are now on your system volume, and you know that since the directory of the storage volume is empty, there is ample room on it for new files.

## MULTIPLE FILE OPERATIONS

You often have occasion to perform the same utility operation on several files. For example, you may copy from one volume to another all files with the file type .MAC, or you may erase from a volume all files with the name TEST. Rather than perform the required operation on the files one at a time, it is easier to use the wildcard construction, a shorthand method provided by the RT-11 operating system. Wildcard construction allows you to substitute an asterisk (\*) or percent sign (%) for a portion of the file name that is variable among all the files you want used

in the operation. For example, specifying DECIND.\* in a command causes the operation to act on all files with the file name DECIND, regardless of their file type; \*.BAK causes the system to act on files with the file type BAK, regardless of their file name. Specifying TEST%.FOR causes the operation to act on all files having a type of FOR, starting with the four characters TEST, and having any fifth character (for example, TESTA.FOR, TEST1.FOR, etc.).

A special use of the wildcard construction involves substitution of an asterisk for both file name and file type. \*.\* implies that all files, regardless of the file name or file type, are to be used in the operation.

Exercises in this chapter and throughout the remainder of the manual demonstrate various uses of the wildcard construction.

Storage volumes provide an area where you can store important files. Since most files are originally created on the default system volume, you must copy them from the system volume to the storage volume. The following exercises show you how to make backup copies on your storage volume of the two provided demonstration programs (DEMOF1.FOR and DEMOX1.MAC), and how to copy to the storage volume the two programs you created (GRAPH.FOR and SUM.MAC).

The monitor command that copies files between volumes is the COPY command. This command instructs the system to duplicate the file that you indicate as input, and then gives the new file the name and file type that you specify as output. The original version of the file is unaffected; that is, a copy of the original version is made and moved to the new volume.

To copy GRAPH.FOR to your storage volume under the new name GRAPH.TWO, type:

### Long Command Format

```
.COPY␣
From? GRAPH.FOR␣
To ? VOL:GRAPH.TWO␣
```

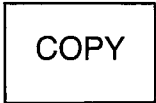
(System volume is assumed for input.)

### Short Command Format

```
.COPY GRAPH.FOR VOL:GRAPH.TWO␣
```

The system makes a copy of the file GRAPH.FOR on the storage volume and gives the copy the name GRAPH.TWO. When the operation is complete, the monitor prints a period at the left margin and waits for you to enter the next command. This time, copy SUM.MAC to the storage volume.

## FILE COPYING OPERATIONS



### Long Command Format

```
.COPY(RET)
From? SUM.MAC(RET)
To ? VOL:SUM.MAC(RET)
```

### Short Command Format

```
.COPY SUM.MAC VOL:SUM.MAC(RET)
```

The system copies the file SUM.MAC to your storage volume and gives the copy the name SUM.MAC.

Now, copy the two provided demonstration programs, DEMOF1.FOR and DEMOX1.MAC, to the storage volume.

### Long Command Format

```
.COPY(RET)
From? DEMOF1.FOR(RET)
To ? VOL:DEMOF1.FOR(RET)
```

```
.COPY(RET)
From? DEMOX1.MAC(RET)
To ? VOL:DEMOX1.MAC(RET)
```

### Short Command Format

```
.COPY DEMOF1.FOR VOL:DEMOF1.FOR(RET)
.COPY DEMOX1.MAC VOL:DEMOX1.MAC(RET)
```

A directory of your storage volume should verify that it now contains these four files.<sup>1</sup>

### Long and Short Command Format

```
.DIRECTORY VOL:(RET)
08-Jan-83
GRAPH .TWO      2  08-Jan-83      SUM  .MAC 3  08-Jan-83
DEMOF1.FOR     2  26-Aug-82      DEMOX1.MAC 3  26-Aug-82
4 Files, 10 Blocks
4752 Free blocks
```

## FILE RENAMING OPERATIONS

The directory you just listed shows that you copied the GRAPH demonstration file to your storage volume under a new file type, .TWO. Assume you did not intend to copy it using a new file type and now wish that it were assigned its original file

<sup>1</sup>If you are using magtape as your storage volume, read the section entitled Directory vs Nondirectory-Structured Volumes in Appendix B.

type, .FOR. Use the monitor RENAME command to rename the file already on the storage volume.<sup>1</sup>

RENAME

### Long Command Format

```
.RENAME (RET)
From? VOL:GRAPH.TWO (RET)
To ? VOL:GRAPH.FOR (RET)
```

### Short Command Format

```
.RENAME VOL:GRAPH.TWO VOL:GRAPH.FOR (RET)
```

The RENAME command simply changes the file name or file type of a file in the volume directory without altering or moving the file itself. When you perform a rename operation, the volume indicated in the input and output portions of the command must be the same; otherwise a system message is printed.

Rename the file copies DEMOX1.MAC and DEMOF1.FOR presently on your storage volume to EXAMP.MAC and EXAMP.FOR respectively.

```
.RENAME VOL:DEMOX1.MAC VOL:EXAMP.MAC (RET)
.RENAME VOL:DEMOF1.FOR VOL:EXAMP.FOR (RET)
```

Again list a directory of your storage volume to verify that the renaming operation occurred.

### Long and Short Command Format

```
.DIRECTORY VOL: (RET)
08-Jan-83
GRAPH.FOR      2 08-Jan-83      SUM .MAC      3 08-Jan-83
EXAMP.FOR      2 26-Aug-82      EXAMP.MAC    3 26-Aug-82
4 Files, 10 Blocks
4752 Free blocks
```

Once copies of your important files are stored on a storage volume, you can delete from the system volume — or any other volume — those files that you no longer need. The file deletion operation removes information about the file from the volume's directory; the space that the file occupies on the volume becomes available for reuse. Files that you want to delete generally include .BAK files created during editing, temporary files created by utility programs, or any other unnecessary files.

## FILE DELETION OPERATIONS

<sup>1</sup>Magtape users cannot use the RENAME command and should read the section entitled Alternate Rename Operation for Magtape Users in Appendix B.

Now that you have copies of your important files, you can delete several files from your system volume. For example, you can delete all files with a .BAK file type created as a result of editing. You can delete the file DECIND.USA, since this was created only for editing practice. Finally, you can delete the files GRAPH.FOR and SUM.MAC, since copies of these are now on VOL:.

Do not delete DEMOF1.FOR and DEMOX1.MAC from your system volume, even though copies of these are also on VOL:. You should consider these two files as part of the RT-11 operating system, and therefore should not erase them from the system volume. These copies can serve as additional backups for the files on the storage volume.

DELETE

The monitor DELETE command is used to delete file names from a volume. You can specify as many as six input files for deletion.

### Long Command Format

```
.DELETE(RET)  
Files? DECIND.USA,GRAPH.FOR,SUM.MAC(RET)
```

### Short Command Format

```
.DELETE DECIND.USA,GRAPH.FOR,SUM.MAC(RET)
```

If wildcard construction is used or if the /QUERY option is specified, the DELETE command requests confirmation from you by printing each file name on the terminal before it deletes it. If you type a Y response, the system deletes the file, while an N response instructs the system to ignore that file and go on to the next. Notice how you use the wildcard construction in the input file to delete all files with a .BAK file type.

### Long Command Format

<pre>.DELETE<sup>(RET)</sup> Files? *.BAK<sup>(RET)</sup> Files deleted: DK:DECIND.BAK ? Y<sup>(RET)</sup> DK:SUM.BAK ? Y<sup>(RET)</sup> DK:GRAPH.BAK ? Y<sup>(RET)</sup></pre>	<pre>.DELETE/QUERY<sup>(RET)</sup> Files? GRAPH.BAK,SUM.BAK,DECIND.BAK<sup>(RET)</sup> Files deleted: DK:DECIND.BAK ? Y<sup>(RET)</sup> DK:SUM.BAK ? Y<sup>(RET)</sup> DK:GRAPH.BAK ? Y<sup>(RET)</sup></pre>
--	---

### Short Command Format

<pre>.DELETE *.BAK<sup>(RET)</sup> Files deleted: DK:DECIND.BAK ? Y<sup>(RET)</sup> DK:SUM.BAK ? Y<sup>(RET)</sup> DK:GRAPH.BAK ? Y<sup>(RET)</sup></pre>	<pre>.DELETE/QUERY GRAPH.BAK,SUM.BAK,DECIND.BAK<sup>(RET)</sup> Files deleted: DK:DECIND.BAK ? Y<sup>(RET)</sup> DK:SUM.BAK ? Y<sup>(RET)</sup> DK:GRAPH.BAK ? Y<sup>(RET)</sup></pre>
---	--

## FILE PROTECTION OPERATIONS

You can give a file a protected status to prevent it from being deleted from the volume it resides on. A file that has a protected status cannot be deleted until that status is removed.

The files DEMOF1.FOR and DEMOX1.MAC should still be on your system volume. These files must not be deleted because they are part of your RT-11 operating system. You can give these two files a protected status to ensure that they are not accidentally deleted.

The monitor PROTECT command is used to give files a protected status. You can specify up to six input files (separated with commas) with this command. As with the DELETE command, if you use the wildcard construction or specify the /QUERY option, the RT-11 system requests confirmation before protecting each file.

PROTECT

Use the PROTECT command to give the files DEMOF1.FOR and DEMOX1.MAC a protected status.

### Long Command Format

```
.PROTECT(RET)  
Files? DEMOF1.FOR,DEMOX1.MAC(RET)
```

### Short Command Format

```
.PROTECT DEMOF1.FOR,DEMOX1.MAC(RET)
```

List a directory of these two files and notice the "P" next to the number of blocks in the second column.

### Long and Short Command Format

```
.DIRECTORY DEMOF1.FOR,DEMOX1.MAC(RET)  
08-Jan-83  
DEMOX1.MAC 3P 26-Aug-82 DEMOF1.FOR 2P 26-Aug-82  
2 Files, 5 Blocks  
921 Free blocks
```

The UNPROTECT command removes a file's protected status so that the file can be deleted. You can determine whether a file is protected or not by listing the directory. If a file is protected, a "P" will appear next to the block size number of the file's directory entry. You can specify up to six input files (separated with commas) with this command. As with the DELETE command, if wildcard construction is used or if the /QUERY option is specified the UNPROTECT command requests confirmation for each file.

UNPROTECT

Remove the protected status from the files DEMOF1.FOR and DEMOX1.MAC.

### Long Command Format

```
,UNPROTECT(RET)  
Files? DEMOF1, FOR , DEMOX1, MAC(RET)
```

### Short Command Format

```
,UNPROTECT DEMOF1, FOR , DEMOX1, MAC(RET)
```

Again list a directory of the two files DEMOF1.FOR and DEMOX1.MAC. The "P"'s are removed from the listing, signifying that the files are no longer protected.

### Long and Short Command Formats

```
,DIRECTORY DEMOF1, FOR , DEMOX1, MAC(RET)  
08-Jan-83  
DEMOX1, MAC 3 26-Aug-82 DEMOF1, FOR 2 26-Aug-82  
2 Files, 5 Blocks  
921 Free blocks
```

## FILE LISTING OPERATIONS

You sometimes need a listing of a file before you can decide whether or not to delete it. In Chapter 5, you used the RT-11 editor to get listings of the files you created. You can also use monitor commands to obtain listings of files. One command lists a file on the console terminal; another lists a file on the line printer.<sup>1</sup> The system volume is the assumed storage volume for the input file.

Type one of the following sets of commands to obtain listings of EXAMP.MAC and EXAMP.FOR.

PRINT

TYPE

### Long Command Format

(Line Printer)

```
,PRINT(RET)  
Files? VOL: EXAMP, MAC(RET)
```

```
,PRINT(RET)  
Files? VOL: EXAMP, FOR(RET)
```

(Terminal)

```
,TYPE(RET)  
Files? VOL: EXAMP, MAC(RET)
```

```
,TYPE(RET)  
Files? VOL: EXAMP, FOR(RET)
```

### Short Command Format

(Line Printer)

```
,PRINT VOL: EXAMP, MAC(RET)
```

```
,PRINT VOL: EXAMP, FOR(RET)
```

(Terminal)

```
,TYPE VOL: EXAMP, MAC(RET)
```

```
,TYPE VOL: EXAMP, FOR(RET)
```

<sup>1</sup>If a line printer is available on your system, you should always use it for listings because of its high speed of printing.

You should perform file maintenance operations periodically as you use the system. File maintenance keeps your system and storage volumes up-to-date and provides maximum free space on volumes for new files.

**COPY**

Copy the specified file from one volume to another.

**DELETE**

Delete the specified file(s) from the volume's directory. Confirmation is required before deleting the file if wildcard construction or the /QUERY option is used.

**DIRECTORY**

List the volume directory on the terminal.

**DIRECTORY/PRINTER**

List the volume directory on the line printer.

**PRINT**

List the contents of the specified file on the line printer.

**PROTECT**

Give the specified file protected status so it cannot be deleted until that status is removed. Confirmation is required before protecting the file if wildcard construction or the /QUERY option is used.

**RENAME**

Give a new name to the specified file.

**TYPE**

List the contents of the specified file on the terminal.

**UNPROTECT**

Remove the protected status of the specified file so that the file can be deleted. Confirmation is required before removing each file's protection if wildcard construction or the /QUERY option is used.

**SUMMARY:  
FILE MAINTENANCE  
COMMANDS**

*RT-11 System User's Guide (AA-5279C-TC)*. Maynard, Mass.: Digital Equipment Corporation, 1983.

**REFERENCE**

A guide to the use of the RT-11 operating system.

)

1

)

)

)

)

## CHAPTER 8 CHOOSING A PROGRAMMING LANGUAGE

Programming languages and language processors are aids provided by the operating system to help you develop programs of your own. Whenever you plan to write a program, you must first decide on the programming language that you will use, since most computer systems support several. After you have chosen the language, you must design and code your program using appropriate language statements and being careful to follow formatting rules and restrictions. Finally, you must use the corresponding language processor, which is stored on the system volume or on a volume of its own, to convert your program statements into a format suitable for execution.

Hundreds of programming languages have been developed for computer systems. Some languages can be used only for specific applications or with a particular computer system. Other languages are general purpose; they are suitable for a variety of problem-solving situations and, in addition, are easy to learn and use. The languages demonstrated in this manual include two well-known and widely used high-level programming languages (BASIC-11 and FORTRAN IV) and one RT-11 system-specific machine-level programming language (MACRO-11).

### HIGH-LEVEL VS MACHINE-LEVEL LANGUAGES

High-level languages, like BASIC-11 and FORTRAN IV, are usually easy to learn and use. You write programs using language statements that need not deal with the specifics of the computer system. The language processor — and perhaps other utility programs as well — handles all conversions that are necessary for program execution. Since a single high-level language statement may perform several computer operations, and since you need not be concerned or familiar with the structure of the computer and peripheral devices, you can concentrate solely on solving the problem at hand. The language processor takes care of translating the statements into computer information.

Thus, high-level languages are considered machine-independent languages because language statements are such that any program written in the language can usually be executed on an entirely different computer system (that supports the language) with few, if any, modifications.

Machine-level languages, on the other hand, such as the assembly language MACRO-11, require that you know about the computer and the peripheral devices and how they work together. You write programs in formats that are closer to those

required for execution. Since a single machine-level language statement usually performs only one computer operation, you must account in your program for each computer operation that will be required.

For this reason, machine-level languages are machine-dependent languages. The program is coded in a format that is not usually interchangeable among systems. Machine-level language programs can be efficient because the knowledgeable programmer will choose the fastest and most precise instructions for getting a job done.

Table 8-1 lists a comparison of high-level vs machine-level languages.

**Table 8-1 Language Comparisons**

<b>High-Level</b>	<b>Machine-Level</b>
Easy to learn and use; no experience required	More difficult to learn and use; familiarity with the computer system required
Machine-independent	Machine-dependent
Many hidden conversions necessary for program execution; more computer memory is used	Only direct translation is necessary for program execution; less computer memory is used
Slower execution time	Faster execution time
Less efficient; the system makes decisions concerning computer operations	More efficient; the programmer makes decisions concerning computer operations
Easier to debug (find and fix errors)	Harder to debug (find and fix errors)
Easier to understand programs; functions added with less difficulty	Harder to understand programs; functions added with greater difficulty

Beginning programmers, students, commercial applications programmers, and the casual computer user prefer high-level languages because they are less difficult to learn and to use, and they produce fast results. System programmers, on the other hand, may prefer machine-level languages for writing programs (those that make up an operating system, for example) that must often be as fast, efficient, and concise as possible.

**RT-11  
PROGRAMMING  
LANGUAGES**

The designers of a computer system generally select programming languages that will satisfy and suit the current (or perhaps potential) system user environment. The RT-11 computer system is designed for use in many environments: education, business, laboratory, etc. Some of its applications include data acquisition and analysis, record keeping, control systems, and learning through computer simulation. RT-11 programmers and users include both the very knowledgeable and the student/beginner.

To satisfy the varied requirements of these environments, RT-11 supports several programming languages:

High-Level	Machine-Level
BASIC-11	MACRO-11
FORTRAN IV	
DIBOL	

Whenever you choose one or more of these programming languages for your own use, consider the following criteria:

- What is your programming experience? What languages do you already know?
- How much time do you have to learn a new language?
- For what applications will you use the language? How important are program speed and efficiency?
- Will you use your program on any other computer systems?

If you are already familiar with a language supported by the system, you may prefer to continue using that language rather than spend time learning a new one. However, if you want to learn a language, consider your application. High-level languages handle most programming jobs. Unless you plan to write extremely detailed or time-critical programs, you should select a high-level language.

If you are a beginning programmer, you may prefer to start with a language like BASIC-11, which is a conversational, interactive language. Language statements use simple, English-like words and common mathematical expressions. You can request immediate answers to problems by using the immediate modes of the language, or you can create detailed programs by combining single language statements into larger segments. BASIC-11 is a superset of the industry-standard BASIC developed at Dartmouth College. Chapter 10 of this manual describes BASIC-11 in more detail.

RT-11 FORTRAN IV is a superset of the industry-standard FORTRAN IV. This language has long been recognized for its use in the scientific field; in addition, it is one of the most commonly supported languages across systems. You may decide to choose FORTRAN IV because it is a more powerful language than BASIC-11 or because you plan to use your programs on more than one system. Chapter 9 of this manual describes FORTRAN IV in more detail.

Finally, if you are an experienced user, you may select the machine-level programming language MACRO-11. This is a powerful language that allows user programs to access and utilize every possible feature available on the RT-11 computer system. The language requires a considerable amount of computer experience and knowledge to be used effectively, however. The MACRO-11 language is best for you if you are a system programmer or an experienced high-level language programmer. It is described in more detail in Chapter 11 of this manual.

## **CHOOSING A LANGUAGE FOR THE DEMONSTRATION**

Three RT-11 programming languages are demonstrated in the next several chapters of this manual; FORTRAN IV, BASIC-11, and MACRO-11. Consider your ability as a programmer. If you are a beginner, BASIC-11 is probably the best language for you to start with; FORTRAN IV is also a good choice. However, you need not be proficient in any of these programming languages to perform the exercises provided in this manual.

Your particular RT-11 computer system may not provide all three languages. First check question 10 in the Hardware Configuration section of Chapter 2 to find out which languages are available on your system.

Then select a language to continue the demonstration. If you choose FORTRAN IV, continue to Chapter 9. If you choose BASIC-11, go on to Chapter 10. If you choose MACRO-11, go to Chapter 11.

## **REFERENCES**

*Digital Equipment Corporation Reference Service, Volume 2: Software and Services.* Maynard, Mass.: Digital Equipment Corporation, 1982.

An overview of the available PDP-11 family products and services.

Katzan, Harry Jr., *Information Technology, The Human Use of Computers.* New York: Mason & Lipscomb Publishers Petrocelli Books, 1974.

A textbook covering basic computing concepts, programming languages, and topics in computers and society. See Part II, Chapters, 7, 8, and 9.

*PDP-11 Software Handbook* (EB-21759-20). Maynard, Mass.: Digital Equipment Corporation, 1982-83.

A general overview and introduction to available PDP-11 software, operating systems, and language processors.

## CHAPTER 9 RUNNING A FORTRAN IV PROGRAM

The FORTRAN IV programming language<sup>1</sup> is a machine-independent programming language that was originally designed as a quick and easy aid for solving mathematical equations and formulas. However, FORTRAN IV is a powerful language and not difficult to learn or use, and is also well suited to many other kinds of applications.

FORTRAN (FORmula TRANslation) IV is an algebraically oriented language. You write a FORTRAN IV program as a sequence of language statements that combine common English words with quasi-algebraic formulas. You then arrange groups of the language statements into logical units called program units. One or more program units make up an entire executable FORTRAN IV source program.

When you are satisfied with the logic of your FORTRAN IV source program, you use the RT-11 editor to create it as a file (see Chapter 5). You use tabs and spaces to format each line properly, and you may choose to insert comment statements throughout the source code to explain what various parts of the program are doing. When you have finished creating the program as a complete, edited file, you next enter it as input to the FORTRAN IV language processor, which is stored on your system volume or on a separate volume of its own. The FORTRAN IV language processor processes (compiles) the language statements, converting them into internal machine-language code called object code. This code is next processed by the system linker, which combines your program units and necessary system-supplied routines to make your program suitable for execution. The development of an executable FORTRAN IV program is represented in Figure 9-1.



**Figure 9-1 Evolution of a FORTRAN IV Program**

The FORTRAN IV language processor is a compiler that translates your source program into a machine language program.

<sup>1</sup>The PDP-11 FORTRAN IV programming language conforms to the specifications for American National Standard FORTRAN X3.9-1966.

**DEVELOPING AN  
EXECUTABLE  
FORTRAN IV  
PROGRAM**

**USING THE  
FORTRAN IV  
LANGUAGE  
PROCESSOR**

Since you create a FORTRAN IV source program in ASCII format, you must next translate the program into a machine format that the computer can use. The FORTRAN IV compiler performs the translation, producing as output a new version of the program, called an object module. You may instruct the FORTRAN IV compiler to produce a listing of the source program at the same time. Figure 9-2 is a diagram of the compiler's function.

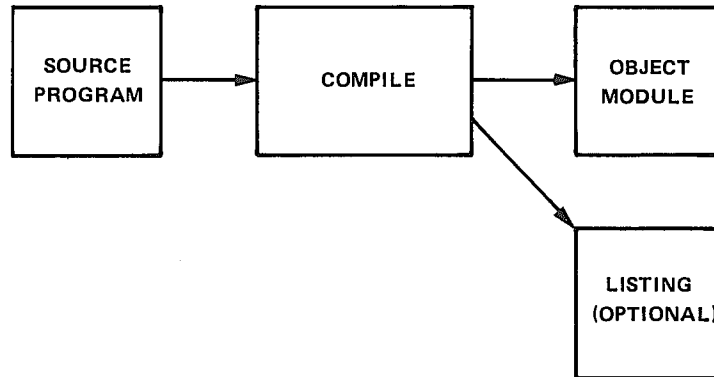


Figure 9-2 Function of a FORTRAN IV Compiler

## USING LIBRARY MODULES

Typical FORTRAN IV programs often require similar operations. Most programs, for example, use routines and instructions that calculate square roots, exponentials, and other arithmetic functions; handle input and output operations; detect certain kinds of error conditions; test values; calculate subscripts; and perform conversions. These commonly used operations have been gathered into a special file called SYSLIB.OBJ (default System Library), which is provided with the RT-11 operating system and is stored on your system volume.

During the processing of your source program, the FORTRAN IV compiler examines each language statement in the program. If you use operations that are provided in SYSLIB, the compiler notes them and makes references to SYSLIB. The compiler translates all the information gathered during processing (your converted language statements and the references to SYSLIB) into numerical data called object code that the system linker can use. The result of the compilation, therefore, is an object format file, called an object module, which is automatically joined with SYSLIB (containing many object modules) and with any other required object modules at link time. Linking all the necessary object modules together produces a complete, workable FORTRAN IV program.

The FORTRAN IV object time system (OTS) is also needed to successfully compile a FORTRAN IV program; this system is in

the file FORLIB.OBJ. Whether this file is included in SYSLIB or not depends on how your system was built.

To link the example FORTRAN IV program, you must either include FORLIB.OBJ in SYSLIB, or specify SYSLIB before FORLIB in the link command. For instructions on how to include FORLIB in SYSLIB, refer to Section 3.4 in the *RT-11 FORTRAN IV Installation Guide*.

In Chapter 5 you used the RT-11 editor to create a FORTRAN IV source program, which you then stored on your storage volume. Since a source program is in ASCII format, the next step is to use the FORTRAN IV compiler to convert it to object code.

Some RT-11 systems store the FORTRAN IV compiler on a volume apart from the system volume.<sup>1</sup> You can quickly determine whether the FORTRAN IV compiler is on your system volume by using the DIRECTORY command.

```
.DIRECTORY SY:FORTRA.SAV(RET)
```

In the directory listing that results, if the directory entry for FORTRA.SAV is included, then the required FORTRAN IV files are on your system volume. If, however, FORTRA.SAV did not appear in the directory listing, then the required files are not part of your system volume. Before you can use the compiler, you must make a volume substitution. Read the section in Appendix B entitled Using the FORTRAN/BASIC Language Volume.

The next step involves using the monitor COPY command to copy the FORTRAN IV source program from the storage volume (where you stored it in Chapter 7) back to the system volume, which serves as the default volume for input/output operations.

Remember that on your storage volume are two FORTRAN IV source programs, the one you created (GRAPH.FOR) and the one provided as part of the system (EXAMP.FOR). Which of these you should use depends on the results of the source comparison you performed in Chapter 6. If the comparison resulted in no differences except for the title lines, copy your own program (GRAPH.FOR) as follows:

### Long Command Format

```
.COPY(RET)
From? VOL:GRAPH.FOR(RET)
To ? GRAPH.FOR(RET)
```

<sup>1</sup>This is true for any RT-11 system volume that does not have enough free blocks to accommodate the FORTRAN IV system files. RX01 diskette is an example.

## COMPILING THE FORTRAN IV PROGRAM

### Short Command Format

```
.COPY VOL:GRAPH.FOR GRAPH.FORRET
```

However, if differences were printed in addition to the title lines, use the provided program (EXAMP.FOR) instead, copying it under the new name GRAPH.FOR:

### Long Command Format

```
.COPYRET  
From? VOL:EXAMP.FORRET  
To ? GRAPH.FORRET
```

### Short Command Format

```
.COPY VOL:EXAMP.FOR GRAPH.FORRET
```

The FORTRAN IV source file now resides on your system volume under the name GRAPH.FOR and is the file that you will process with the FORTRAN IV compiler. The command used to compile a FORTRAN IV source program is the monitor FORTRAN command.

FORTRAN

Use the FORTRAN command with its /LIST option to compile your program and produce a listing. The system prompt asks you to supply the input file name. You can omit typing the .FOR file type since the FORTRAN command assumes this file type unless you indicate otherwise. The system will assign the name GRAPH.OBJ to the object module and GRAPH.LST to the listing file and store both newly created files on your system volume, which is the default storage volume for input/output operations.

### Long Command Format

```
.FORTRANRET  
Files? GRAPH/LISTRET
```

### Short Command Format

```
.FORTRAN GRAPH/LISTRET
```

Compilation begins. If the compiler discovers an error during processing, it prints a message. In this particular case, you should see the following on your terminal printer or screen:

```
.MAIN,  
?FORTRAN-I-[.MAIN.] Errors: 5, Warnings: 0  
FUN  
?FORTRAN-I-[FUN ] Errors: 1, Warnings: 0
```

This indicates that, during processing, the FORTRAN IV compiler found six errors in the source program. It helps at this

point to look at the listing produced by the compiler, because more information is shown there. Print the listing on either the line printer or terminal, using one of the following commands:

### Long Command Format

(Line printer)

```
.PRINTRET
Files? GRAPH.LSTRET
```

(Terminal)

```
.TYPERET
Files? GRAPH.LSTRET
```

### Short Command Format

(Line printer)

```
.PRINT GRAPH.LSTRET
```

(Terminal)

```
.TYPE GRAPH.LSTRET
```

Your listing should look like the following example.

### NOTE

You do not need to understand the FORTRAN IV language or the way this program works to successfully complete the exercises in this chapter.

```
FORTRAN IV      V02.5   Sat 08-Jan-83 15:52:04      PAGE 001

C GRAPH.FOR      VERSION 1
C THIS PROGRAM PRODUCES A PLOT ON THE TERMINAL
C OF AN EXTERNAL FUNCTION, FUN(X,Y)
C THE LIMITS OF THE PLOT ARE DETERMINED BY THE DATA STATEMENTS
C "STAB" IS FILLED WITH A TABLE OF HEIGHT FLAGS
C "STRING" IS USED TO BUILD A LINE OF GRAPH FOR PRINTING
0001      SCAL (ZMIN,ZMAX,MAXZ,K)=ZMIN+FLOAT(K-1)*(ZMAX-ZMIN)/FLOAT(MAXZ-1)
0002      LOGICAL*1 STRING(13,3),STAB(100)
0003      DATA XMIN,XMAX,MAXX/-5,5,45/
0004      DATA YMIN,YMAX,MAXY/-5,5,72/
0005      DATA FMIN,FMAX/0.0,1.0/
0006      CALL SCOPY(' - 1 2 3 4 5 6 7 8 9 +',STAB)
0007      MAXF=LEN(STAB)
0008      DO 20 IX=1,MAXX
0009          X=SCAL(XMIN,XMAX,MAXX,IX)
0010          CALL REPEAT('*',STRING,MAXY)
0011          IF(IX.EQ.1 .OR. IX.EQ.MAXX) GOTO 20
0013          DO 10 IY=2,MAXY-1
0014              Y=SCAL(YMIN,YMAX,MAXY,IY)
0015              IFUN=2+INT(FLOAT(MAXF-3)*(FUN(X,Y)-FMIN)/(FMAX-FMIN))
0016 10      STRING(IY)=STAB(MINO(MAXF,MAXO(1,IFUN)))
0017 30      CALL PUTSTR(7,STRING,' ')
0018      CALL EXIT
0019      END
FORTRAN IV      Diagnostics for Program Unit .MAIN.
```

```
In line 0003, Error: Modes of variable "XMIN" and data item differ
In line 0004, Error: Modes of variable "YMIN" and data item differ
In line 0008, Error: Reference to undefined statement label
In line 0012, Error: Reference to undefined statement label
In line 0016, Error: Wrong number of subscripts for array "STRING"
```

```
FORTRAN IV      Storage Map for Program Unit .MAIN.
```

```
Local Variables, .PSECT $DATA, Size = 000334 ( 110. words)
```

Name	Type	Offset	Name	Type	Offset	Name	Type	Offset
FMAX	R*4	000230	FMIN	R*4	000224	IFUN	I*2	000312
IX	I*2	000274	IY	I*2	000300	K	I*2	000256
MAXF	I*2	000260	MAXX	I*2	000272	MAXY	I*2	000276
MAXZ	I*2	000254	MAXO	I*2	000316	MINO	I*2	000314

## Running a FORTRAN IV Program

```

X      R*4  000262      XMAX  R*4  000266      XMIN  R*4  000214
Y      R*4  000302      YMAX  R*4  000306      YMIN  R*4  000220
ZMAX   R*4  000250      ZMIN  R*4  000244

```

Local and COMMON Arrays:

```

Name    Type  Section Offset  ---Size--- Dimensions
STAB   L*1   $DATA  000047  000144 ( 50.) (100)
STRING L*1 Vec $DATA  000000  000047 ( 20.) (13,3)

```

Subroutines, Functions, Statement and Processor-Defined Functions:

```

Name    Type  Name    Type  Name    Type  Name    Type  Name    Type
EXIT    R*4    FLDAT   R*4    FUN     R*4    INT     I*2    LEN     I*2
PUTSTR  R*4    REPEAT  R*4    SCAL    R*4    SCDPY   R*4
FORTRAN IV  V02.5  Sat 08-Jan-83 15:52:07  PAGE 001

```

```

0001      FUNCTION FUN(X,Y)
0002      R=SQR(X**2+Y**2)
0003      FUN=X*Y*R*EXP(-R)**2
**** P
0004      RETURN
0005      END
FORTRAN IV  Diagnostics for Program Unit FUN

```

```

In line 0003, Error: [See source listing]
FORTRAN IV  Storage Map for Program Unit FUN

```

Local Variables, PSECT \$DATA, Size = 000020 ( 8. words)

```

Name    Type  Offset  Name    Type  Offset  Name    Type  Offset
FUN     R*4    000004  Equ R   R*4    000010  X      R*4 @   000000
Y      R*4 @ 000002

```

Subroutines, Functions, Statement and Processor-Defined Functions:

```

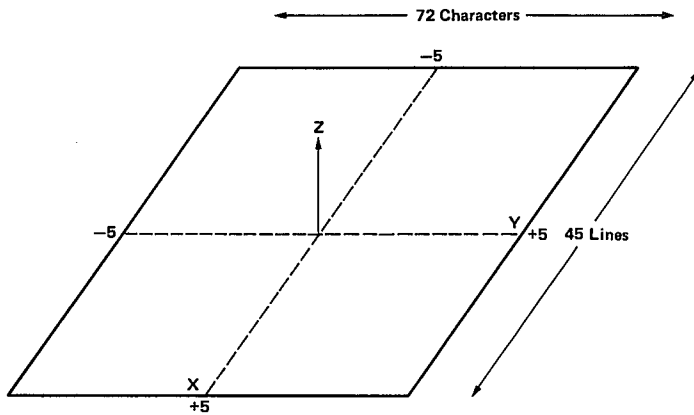
Name    Type  Name    Type  Name    Type  Name    Type  Name    Type
SQRT    R*4

```

The first part of the listing shows the main program unit and consists of the language statements up to, but not including, the function. This is followed by a diagnostics list, then by a storage map. Next the language statements composing the function program unit are listed, again followed by a diagnostics list and a storage map.

Before considering the individual sections of the program listing, first examine the program logic to determine what this program should do. The first few lines of this program are user comment lines that briefly describe the program. Essentially, this program produces on the terminal a graph of a "three-dimensional" function, FUN(X, Y). The graph is plotted using 45 lines down and 72 characters across the terminal page. The limits of the X and Y axes are +5 and -5. The third dimension, height, is a real number within the range 0 to 1 and is represented in the listing as a number within a scale of 1 to 9. These dimensions are illustrated in Figure 9-3.

The SCAL function determines the value of the next coordinate on the graph. The statements within the DO loops calculate the coordinates using the SCAL function and determine the height value. This is done for an entire line of coordinates across the terminal page. The entire line is then printed on the terminal, using the CALL PUTSTR statement; the number 7 in this statement is the FORTRAN IV method of naming the terminal



**Figure 9-3 Dimensions of FUN(X,Y)**

as the output device. This procedure is repeated until all 45 lines of the graph have been printed.

The function to be plotted is shown in the last few lines of the program. It is compiled as a separate program unit and you can edit these lines to plot any function of your choice (several alternate functions are suggested later in the chapter).

This program as it stands contains errors. The compiler detected certain error conditions during processing that prevent the program from working properly. The compiler printed appropriate messages in the diagnostics sections of the program listing.<sup>1</sup> Look first at the messages following the main program unit. Errors were discovered in lines 3, 4, 8, 12, and 16.

The messages for lines 3 and 4 indicate that the floating-point variables "XMIN" and "YMIN" are assigned integer values. The DATA statements must be changed. (The same error exists for "XMAX" and "YMAX"; the compiler, however, lists only the first error that it discovers in a line. Both "MAXX" and "MAXY" are integer variable names, so no error exists for them.) You must correct the DATA statements (lines 3 and 4), then, as follows:

```
DATA XMIN,XMAX,MAXX/-5.0,5.0,45/
DATA YMIN,YMAX,MAXY/-5.0,5.0,72/
```

The next two messages in the diagnostics section show that reference has been made from both lines 8 and 12 to an undefined label. (Line 12 is actually the second portion of line 11, the GO TO statement.) Statement label 20 is referenced in each case, but only labels 10 and 30 are shown in the program. This indicates either that a statement is missing or that a typing

<sup>1</sup>Refer to the *RT-11 System Message Manual* for greater detail about any system messages printed.

error exists. Examination of the program logic shows a typing error in line 17. Label 30 should actually be 20. Correct line 17 as follows:

```
20      CALL PUTSTR(7,STRING,' ')
```

The last message in this diagnostics section states that an incorrect number of subscripts was given for the array "STRING". Again, examination of program logic shows that the error is actually in line 1. "STRING" is really a vector (a one-dimension array), not a matrix (a two-dimension array). Thus the comma is a typing error and line 2 should be changed as follows:

```
LOGICAL*1 STRING(133),STAB(100)
```

Skip next to the diagnostics section for the FUN program unit. The message printed there refers you to the source listing, to line 3. A letter "P" appears next to this line. The *RT-11 System Message Manual* describes a *P* error as an indication of unbalanced parentheses. Notice that the parentheses are not properly matched in this line. Thus, line 3 should be corrected as follows:

```
FUN=(X*Y*R*EXP(-R))**2
```

This explains the errors flagged by the compiler in the diagnostics sections. Other sections of the program listing (storage map, for example) simply provide additional information that is helpful to programmers who wish to check the data types of various symbols and later make sure that object modules have been appropriately linked.

Before you can continue the exercises in this chapter, you must edit, in the source program, those statements that contain errors. If necessary, review the editing commands in Chapter 5. Then use the RT-11 editor to edit the file GRAPH.FOR on your system volume so that the five lines are error-free. Do not rename the file. When you are ready, recompile the program, using the FORTRAN command, and obtain a new object module and a new listing. This time the program should compile without error (that is, no diagnostics should list). The compiler will indicate two warnings, but you can ignore them. If diagnostics occur, you have not edited the program correctly. Compare listings and try to correct your errors, or go back to the beginning of this chapter and repeat the demonstration.

## LINKING OBJECT MODULES TOGETHER

The object module produced by the FORTRAN command is in itself incomplete. As mentioned earlier, it needs parts of the

system library, SYSLIB, and perhaps other object modules and libraries as well, to form a complete functioning program.<sup>1</sup> All required object modules must be joined, or linked together, before the program can work.

Even if your program does not require any other object modules, you must still link it. In addition to joining object modules together, the link operation adjusts the object code to account for many program units being placed one after the other. The result of the link operation is a memory image load module, which is actually a picture of what computer memory looks like just before program execution. Figure 9-4 is a diagram of the link operation.

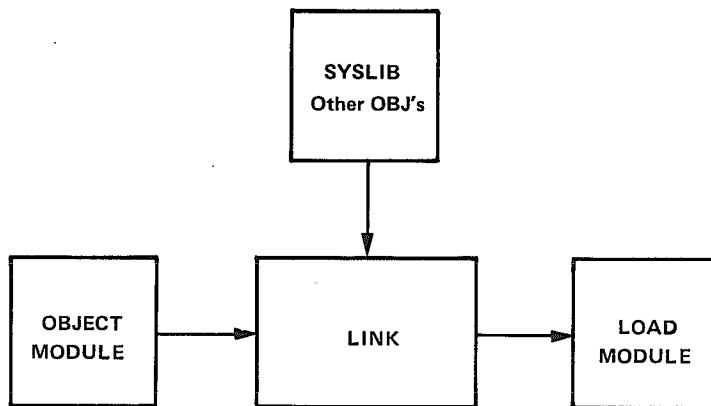


Figure 9-4 The Link Operation

To link the object modules, use the monitor LINK command. The system prompts you to enter the names of the input modules and any libraries other than the system library to be joined together. You can omit typing the .OBJ file types in the command line, since the LINK command assumes this file type for input. The system automatically assigns the file name of the first input file and a file type of .SAV to the output file. The linker will scan the SYSLIB library if it is present on the system volume.

Some RT-11 systems store the linker (LINK.SAV) and the default system library (SYSLIB.OBJ) on a volume apart from the

LINK

<sup>1</sup>For more information on linking files and using library files, see Chapters 12 and 13, respectively.

system volume or the FORTRAN/BASIC language volume.<sup>1</sup> You can quickly determine whether the system library is on your system volume by using the DIRECTORY command.

```
.DIRECTORY SY:SYSLIB.OBJ␣
```

If SYSLIB.OBJ did not appear in the directory listing on your terminal, the required files are not part of your system volume. Before you can link GRAPH.OBJ, you must make a volume substitution. Read the section entitled Using the LINK Volume in Appendix B.

If you have not included the FORTRAN IV library FORLIB.OBJ in SYSLIB, use the DIRECTORY command to see if the library is on your system volume. Type:

```
.DIRECTORY SY:FORLIB.OBJ␣
```

If FORLIB.OBJ did not appear in the directory, the required files are not part of your system volume. Before you can link GRAPH.OBJ, you must make a volume substitution. Read the section entitled Using the LINK Volume in Appendix B.

### **Long Command Format**

FORLIB not included in SYSLIB:

```
.LINK␣  
Files? GRAPH,SYSLIB,FORLIB␣
```

FORLIB included in SYSLIB:

```
.LINK␣  
Files? GRAPH␣
```

### **Short Command Format**

FORLIB not included in SYSLIB:

```
.LINK GRAPH,SYSLIB,FORLIB␣
```

FORLIB included in SYSLIB:

```
.LINK GRAPH␣
```

Any messages printed on the terminal identify error conditions discovered by the system during the link operation (for exam-

---

<sup>1</sup>This is true for any RT-11 system volume that does not have enough free blocks to accommodate the files required for linking. The RX01 diskette is an example.

ple, you may not have specified all the object modules that are needed as input). However, assuming that you edited your source program correctly and that it compiled without error, it should also now link without error.

A load module is one that you can run on the system. Unless your program contains logic errors that prevent it from running properly (errors that the system cannot always detect), running the .SAV version of your file should produce the results you intended. However, if logic errors exist within your program, running the program will produce either erroneous results or none at all. If this is the case, you must study the source program, rework it, reedit it, and perform the compile and link operations again.

If your FORTRAN IV program is error-free, running the .SAV version should produce the expected results. In this demonstration, running the GRAPH.SAV file should produce a graph on the terminal printer or screen.

Before you run GRAPH.SAV, you have the option of changing the output device from the terminal printer or screen to the line printer by using the monitor ASSIGN command to assign device names (see Chapter 4, Assigning Logical Names to Devices). If you prefer to print the graph on the line printer, simply assign the logical device name 7 (which is the FORTRAN IV code for the terminal) to the line printer code (LP:). You have designated a new output device without altering the source program. To change the device assignment to the line printer, type:

### Long Command Format

```
.ASSIGN(RET)
Physical device name? LP:(RET)
Logical device name? 7(RET)
```

### Short Command Format

```
.ASSIGN LP: 7(RET)
```

This assignment remains in effect until you deassign the names or reboot the monitor.

Now, to execute the FORTRAN IV demonstration program, use the monitor RUN command. You can omit typing the .SAV file type since it is assumed within the RUN command.

RUN

### Long and Short Command Format

```
.RUN GRAPH(RET)
```

After a brief pause, the graph begins to print on the terminal (or line printer) and should look like the graph shown in Figure 9-5.

```

*****
*      11111111111111111111      11111111111111111111      *
*      11111111111111111111      11111111111111111111      *
*      11111111      111111      111111      1111111111      *
*      11111111      1111      1111      1111111111      *
*      1111111      222222222222      111      111      222222222222      11111111      *
*      111111      22222      2222      111      111      2222      22222      111111*
*      11111      2222      3      22      11      11      22      3      2222      1111*
*      11111      222      3333333333      22      11      11      22      3333333333      222      1111*
*      1111      22      333      333      22      11      11      22      333      333      22      111*
*      1111      222      333      4444      33      2      1      1      2      33      4444      333      222      111*
*      1111      222      33      4444444      3      2      11      11      2      3      4444444      33      222      111*
*      1111      222      33      4444      444      33      2      11      11      2      33      444      4444      33      222      111*
*      1111      222      33      44444444      33      2      11      11      2      33      44444444      33      222      111*
*      11111      222      33      44444444      33      2      11      11      2      33      44444444      33      222      111*
*      111111      222      33      444      33      2      11      11      2      3      444      33      222      1111*
*      1111      22      3333      333      2      1      1      2      333      3333      22      1111      *
*      11111      222      22222222      111      11      22      222      11111      *
*      111111      2222222222      111      111      2222222222      111111      *
*      11111111      1111      1111      1111      11111111      *
*      1111      1111      1111      1111      *
*      1111      1111      *
*      1111      1111      *
*      11111111      1111      1111      11111111      *
*      11111      2222222222      111      111      2222222222      11111      *
*      111111      222      222      111      11      22      222      11111      *
*      1111      22      3333      333      2      1      1      2      333      3333      22      1111      *
*      111111      222      33      444      33      2      11      11      2      3      444      33      222      1111*
*      11111      222      33      44444444      33      2      11      11      2      33      44444444      33      222      111*
*      1111      222      33      4444      444      33      2      11      11      2      3      444      4444      33      222      111*
*      1111      222      33      44444444      33      2      11      11      2      33      444      4444      33      222      111*
*      1111      222      333      4444      33      2      1      1      2      33      4444      333      222      111*
*      1111      22      333      333      22      11      11      22      333      333      22      111*
*      11111      222      3333333333      22      11      11      22      3333333333      222      111*
*      11111      2222      3      22      11      11      22      3      2222      1111*
*      111111      22222      2222      111      111      2222      22222      11111*
*      1111111      222222222222      111      111      222222222222      111111      *
*      11111111      1111      1111      11111111      *
*      11111111111111111111      11111111111111111111      *
*      11111111111111111111      11111111111111111111      *
*****

```

Figure 9-5 The Result of GRAPH.SAV

**COMBINING OPERATIONS**

**EXECUTE**

To produce these results, you first compiled the FORTRAN IV source program (GRAPH.FOR), then linked it with the default library (SYSLIB.OBJ), and finally ran the resulting .SAV file (GRAPH.SAV). You can combine these three operations using one monitor command, the EXECUTE command.

**NOTE**

The use of the EXECUTE command requires the following files on your system volume:

- FORTRA.SAV      LINK.SAV      GRAPH.FOR
- SYSLIB.OBJ      FORLIB.OBJ (if not included in SYSLIB)

If you have substituted the special LINK volume for your system volume, you do not have the necessary files to use the EXECUTE command. Proceed to the next section, entitled Alternate Functions.

The EXECUTE command instructs the system to select the language processor, then process, link, and run the program. There are several ways to establish which language processor the EXECUTE command invokes. One way is to specify a language-name option, such as /MACRO, which invokes the MACRO assembler. Another way is to omit the language-name option and 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 GRAPH.FOR, for example, invokes the FORTRAN IV compiler. A third way to establish the language processor is to let the system choose a file type of .MAC, .DBL, or .FOR for the source file you name. If, for example, you specify the file GRAPH, the monitor searches device SY: (your system device) for the files GRAPH.MAC, GRAPH.DBL, and GRAPH.FOR, in that order. If the monitor finds neither GRAPH.MAC nor GRAPH.DBL, it invokes the FORTRAN IV language processor to compile GRAPH.FOR. For example, to combine the compile-link-run operations that you performed in this chapter, you would use the following command (do not actually type this command until you have read the next section, Alternate Functions):

### Long and Short Command Format

FORLIB not included in SYSLIB:

```
.EXECUTE GRAPH/FORTRAN/LIST/LINKLIB:SYSLIB/LINKLIB:FORLIB
```

FORLIB included in SYSLIB:

```
.EXECUTE GRAPH/FORTRAN/LIST
```

The following are some alternate functions that you can substitute in your FORTRAN IV source program to produce different graphs. Simply reedit the program (GRAPH.FOR) so that lines 1-5 in the function portion at the end contain one of the following alternate functions. Then compile, link, and run the programs as described in the previous sections. If the necessary files are available on your system volume (see the previous section, Combining Operations), use the EXECUTE command to run the program. The source program compiles, links, and runs, and the new graph prints on the terminal (or line printer).

## ALTERNATE FUNCTIONS

### Function 1

```
FUNCTION FUN(X,Y)
FUN=EXP(-SQRT(X**2+Y**2))
RETURN
END
```

### Function 2

```
FUNCTION FUN(X,Y)
R=SQRT(X**2+Y**2)
FUN=X*Y*(R-3.)/(1.+EXP(3.*(R-3.5)))
RETURN
END
```

### Function 3

```
FUNCTION FUN(X,Y)
FUN=EXP(+SQRT(X**2+Y**2))/1177.4
RETURN
END
```

### SUMMARY: COMMANDS TO RUN FORTRAN PROGRAMS

#### EXECUTE

Combine the compile-link-run operations into one command.

#### EXECUTE file

Combine the compile-link-run operations into one command.  
Specify the libraries to be used during linking.

#### EXECUTE file/FORTRAN

Combine the compile-link-run operations into one command,  
and specify the input file to be a FORTRAN file.

#### EXECUTE/LIST

Combine the compile-link-run operations into one command.  
Obtain a listing file of the source program and print on line  
printer.

#### FORTRAN

Compile the FORTRAN IV source program and produce an  
object module.

#### FORTRAN/LIST

Compile the FORTRAN IV source program and produce both  
an object module and a listing file.

#### LINK

Link individual object modules together to form a complete  
program and produce a load module.

#### RUN

Run the indicated load module.

### FILE MAINTENANCE

Before continuing further you should perform the necessary file  
maintenance operations.

#### NOTE

If you used a special LINK volume to perform this dem-  
onstration, turn now to the section entitled FOR-  
TRAN/LINK File Maintenance in Appendix B.

Obtain a directory of all files on your system volume that have  
the name GRAPH regardless of file type; these files were cre-  
ated as a result of the exercises in this chapter.

**Long and Short Command Format**

```
.DIRECTORY GRAPH.*(RET)
08-Jan-83
GRAPH .SAV 21 08-Jan-83    GRAPH .LST 8 08-Jan-83
GRAPH .BAK 2 08-Jan-83    GRAPH .FOR 2 08-Jan-83
GRAPH .OBJ 16 08-Jan-83
5 Files, 49 Blocks
447 Free blocks
```

The fact that you have corrected errors in the source file GRAPH.FOR makes the version of that file on your storage volume obsolete. Therefore, transfer the updated copy from your system volume to VOL:, replacing the copy of GRAPH.FOR on the storage volume with the new version.

**Long Command Format**

```
.COPY(RET)
From? GRAPH.FOR(RET)
To ? VOL:GRAPH.FOR(RET)
```

**Short Command Format**

```
.COPY GRAPH.FOR VOL:GRAPH.FOR(RET)
```

Similarly, transfer GRAPH.LST, GRAPH.OBJ, and GRAPH.SAV to your storage volume. This allows you to examine a listing or rerun the FORTRAN IV program without recompiling and relinking the source.

**Long Command Format**

```
.COPY(RET)
From? GRAPH.LST,GRAPH.OBJ,GRAPH.SAV(RET)
To ? VOL:(RET)
Files copied:
DK:GRAPH.LST    to VOL:GRAPH.LST
DK:GRAPH.OBJ    to VOL:GRAPH.OBJ
DK:GRAPH.SAV    to VOL:GRAPH.SAV
```

**Short Command Format**

```
.COPY GRAPH.LST,GRAPH.OBJ,GRAPH.SAV VOL:(RET)
Files copied:
DK:GRAPH.LST    to VOL:GRAPH.LST
DK:GRAPH.OBJ    to VOL:GRAPH.OBJ
DK:GRAPH.SAV    to VOL:GRAPH.SAV
```

Once you have transferred all files of value to your storage volume, delete the useless files — that is, all the GRAPH files — from the system volume.

### Long Command Format

```
.DELETE(RET)
Files? GRAPH.*(RET)
Files deleted:
DK:GRAPH.BAK ? Y(RET)
DK:GRAPH.SAV ? Y(RET)
DK:GRAPH.FOR ? Y(RET)
DK:GRAPH.LST ? Y(RET)
DK:GRAPH.OBJ ? Y(RET)
```

### Short Command Format

```
.DELETE GRAPH.*(RET)
Files deleted:
DK:GRAPH.BAK ? Y(RET)
DK:GRAPH.SAV ? Y(RET)
DK:GRAPH.FOR ? Y(RET)
DK:GRAPH.LST ? Y(RET)
DK:GRAPH.OBJ ? Y(RET)
```

Finally, obtain an up-to-date directory listing of your storage volume so that you can see its current status.

### Long and Short Command Format

```
.DIRECTORY VOL:(RET)
08-Jan-83
SUM .MAC 3 08-Jan-83 EXAMP .FOR 2 26-Aug-82
EXAMP .MAC 3 26-Aug-82 GRAPH .FOR 2 08-Jan-83
GRAPH .LST 8 08-Jan-83 GRAPH .OBJ 16 08-Jan-83
GRAPH .SAV 21 08-Jan-83
7 Files, 55 blocks
4707 Free blocks
```

This completes the FORTRAN IV demonstration. Continue to Chapter 12 to read about the linking process. If you followed the special instructions in Appendix B to load the language volume, leave this volume in device unit 0 until you have finished Chapter 12.

## REFERENCES

McCracken, Daniel D., *A Simplified Guide to FORTRAN Programming*. New York: Wiley, 1974.

An introduction to programming in the FORTRAN language.

*PDP-11 FORTRAN Language Reference Manual* (AA-1855D-TC). Maynard, Mass.: Digital Equipment Corporation, 1980.

A reference manual and guide to programming in the PDP-11 FORTRAN IV language.

*RT-11 FORTRAN IV Installation Guide* (AA-5240E-TC). Maynard, Mass.: Digital Equipment Corporation, 1980.

An RT-11-specific manual that contains instructions for installing the RT-11 FORTRAN IV language processor, and describes known problems and differences between versions.

*RT-11 RSTS-E FORTRAN IV User's Guide* (AA-5749B-TC). Maynard, Mass.: Digital Equipment Corporation, 1980.

An RT-11-specific manual that contains information necessary to compile, link, run, and debug a FORTRAN IV program.

## CHAPTER 10 RUNNING A BASIC-11 PROGRAM

The BASIC-11 program language<sup>1</sup> is a machine-independent programming language that is one of the easiest languages for the beginning programmer to learn. It has both elementary language features that you use to write simple programs, and more advanced operations that allow you to produce complex and efficient programs. In addition, a special "immediate mode" lets you use BASIC-11 like a calculator to obtain instant answers to mathematical problems.

BASIC (Beginner's All-purpose Symbolic Instruction Code) -11 is conversational in nature. It uses simple English keywords and common mathematical expressions to form easily understood language statements.

You write a BASIC-11 program as a series of one or more program lines. You begin each program line with a number that both identifies the line and indicates the order in which the line will be processed. Individual program lines contain one or more BASIC-11 language statements that define the operations to be performed.

When you are satisfied with the logic of your BASIC-11 source program, you create it as a file. However, unlike your methods under other programming languages, you create the file under the control of the BASIC-11 language processor, which is part of the RT-11 operating system and is stored on your system volume or on a separate volume of its own. Thus, you use commands that are part of the BASIC-11 language processor to create and edit the program, list it, run it, and save it for later use.

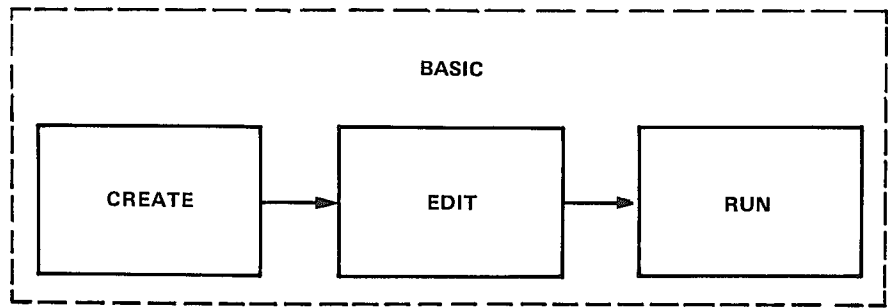
The BASIC-11 language processor is an interactive interpreter. It allows you to create and execute a program in its entirety or a few lines at a time. The interpreter examines each program language statement, interprets it, and executes it before going on to the next. If it discovers an error that prevents further processing, it prints on the terminal a message informing you of the error condition and stops. You correct the error so that execution can continue past that point, and then rerun the program.

---

<sup>1</sup>BASIC-11 is a superset of the standard BASIC language developed at Dartmouth College.

**DEVELOPING  
A BASIC-11  
PROGRAM**

**USING THE  
BASIC-11  
LANGUAGE  
PROCESSOR**



**Figure 10-1 Functions of the BASIC-11 Language Processor**

### USING THE BASIC-11 INTERPRETER

The functions of program creation, editing, processing, and execution are all handled by the BASIC-11 language processor. Some RT-11 systems store the BASIC-11 interpreter (language processor) on a volume apart from the system volume.<sup>1</sup> You can quickly determine whether the BASIC-11 interpreter is on your system volume by typing the monitor DIRECTORY command and specifying the BASIC.SAV program.

```
.DIRECTORY BASIC.SAV(RET)
```

In the directory listing that results, if the directory entry for BASIC.SAV is listed on your terminal, then the required BASIC-11 files are on your system volume and you are ready to use the interpreter. However, if BASIC.SAV did not appear in your listing, then the required files are not part of your system volume. Before you can use the interpreter, you must make a volume substitution. Read the section in Appendix B entitled Using the FORTRAN/BASIC Language Volume.

BASIC

Now use the monitor BASIC command to activate the BASIC-11 interpreter:

#### Long and Short Command Format

```
.BASIC(RET)  
BASIC-11/RT-11 V2.1  
OPTIONAL FUNCTIONS (ALL, NONE, OR INDIVIDUAL)?
```

A prompting message is printed by BASIC-11. You must respond with an A, N, or I and a carriage return to indicate whether you want to preserve all, none, or some of the arithmetic functions initially provided by BASIC-11. BASIC-11's functions include operations that calculate random numbers, deter-

<sup>1</sup>This is true for any RT-11 system volume that does not have enough free blocks to accommodate the BASIC-11 system files. RX01 diskette is an example.

mine absolute values, convert octal and binary numbers to decimal, and so on. You can conserve memory space by saving only those functions that your program needs. For now, however, instruct BASIC-11 to save all the functions.

```
A␣
READY
```

BASIC-11 prints the READY message to indicate that it is ready to accept a BASIC-11 command. Any text that you type that is not preceded by a BASIC-11 command is accepted as program (or immediate mode) input. If at any time you wish to return to the monitor command mode, simply type the BYE command following the READY message. READY appears after any BASIC-11 execution that is completed or interrupted by a double CTRL/C, or after any BASIC-11 wait condition that is terminated by a single CTRL/C.

BYE

#### NOTE

You do not need to understand the BASIC-11 language or the way the examples work to perform successfully the exercises in this chapter.

Immediate mode allows you to use the BASIC-11 interpreter like a calculator to obtain immediate answers to arithmetic problems. You enter the appropriate BASIC-11 statement keyword and any necessary mathematical formula. When you press the carriage return key, BASIC-11 immediately calculates and prints the results. (Use the terminal DELETE key and the CTRL/U command to correct any typing errors.)

#### Immediate Mode

```
PRINT (128+75)*3␣
609
```

BASIC-11 adds the two numbers in parentheses, multiplies them by 3, and prints the answer. The PRINT statement causes the answer to be printed on the terminal. The following command provides another example:

PRINT

```
PRINT INT(34.67)␣
34
READY
```

The greatest integer less than or equal to 34.67 is printed.

You can combine several statements on a single line, or on several lines, including variable names, arithmetic equations, and data. Individual statements are separated from one another by a backslash (\) character. BASIC-11 considers all the

information, calculates the answer and prints it on the terminal, as illustrated in the following example:

```
A=5\B=14\C=,3729(RET)
READY
PRINT "THE HEIGHT IS";A*SIN(C)+B;"METERS"(RET)
THE HEIGHT IS 15,8216 METERS
READY
```

The first statement equates variable names with values; the second statement introduces a formula for calculating a result and prints it.

You can use immediate mode to solve fairly lengthy and complicated mathematical problems by combining statements and printing identifying messages. However, immediate mode information is temporary. You cannot save it, and you can change it only by retyping every statement line. If your needs are more complex, or if you want to save your statements, you should create a BASIC-11 program.

### Creating and Editing a BASIC-11 Program

To create a BASIC-11 program, assign line numbers to language statements and then type the numbered statements on the terminal keyboard.

Now your program lines are saved in memory and you can transfer program control to specific lines within the program, repeat parts of the program any number of times, store the entire program for later use, and perform other similar operations that are not possible in immediate mode.

SUB

Once you have created the program, you use BASIC-11 editing commands to list lines, change lines, add and erase lines, and correct typing errors. In addition to the DELETE key and the CTRL/U command, BASIC-11 provides a SUB command (SUBSTITUTE) for correcting typing errors. This command allows you to substitute new characters for existing ones in a line. For example, type:

```
10 PRINT "THIS IS A BADIC PROGRAM"(RET)
SUB 10 @BAD@BAS@RET)
10 PRINT "THIS IS A BASIC PROGRAM"
READY
```

The SUB command substitutes the letters BAS for BAD in line 10. Use a delimiting character (shown here as @) to separate the old text from the new. The delimiter can be any character as long as it is unique in the line. The corrected line is automatically printed by BASIC-11 after you use the command. As another example, type:

```
15 B=10\C=5(RET)
20 LET A=B+C\PRINT C(RET)
```

There are two typing errors in line 20; the – should be an = and the C at the end of the line should be A. These errors can be corrected with the SUB command, as follows:

```
SUB 20 @-@=@(RET)
20 LET A=B+C \ PRINT C

READY
SUB 20 @C@A@2(RET)
20 LET A=B+C \ PRINT A

READY
```

The second SUB command changes the second occurrence (specified by the 2 after the last @) of C to A.

You can erase an entire line by typing the line number followed by a carriage return:

```
10(RET)
```

You can also use BASIC-11's DEL command<sup>1</sup>. Use the DEL command (DELETE) to erase a single line or several:

DEL

```
DEL 15-20(RET)
```

This erases all statements with line numbers between and including 15 and 20.

To list lines of a program, BASIC-11 provides the LIST command. First, create a few program lines:

LIST

```
5 FOR I=1 TO 10(RET)
20 INPUT J(RET)
25 LET T=T+J(RET)
50 NEXT I(RET)
55 PRINT "THE TOTAL IS";T(RET)
88 END(RET)
```

List individual lines by specifying the line number. For example, type:

```
LIST 5(RET)

NONAME 08-JAN-83 00:18:49

5 FOR I=1 TO 10

READY
```

<sup>1</sup>Do not confuse the BASIC-11 DEL command with the DELETE key on the terminal keyboard.

LISTNH

Notice that BASIC-11 prints a header line. Since you have not as yet assigned a name to your program, BASIC-11 assigns it the name NONAME and prints this name, along with the date (which is only correct if previously entered via the DATE monitor command) and the time when you use the LIST command. You can omit the header line by using the LISTNH command instead of the LIST command.

```
LISTNH 50-88(RET)  
50 NEXT I  
55 PRINT "THE TOTAL IS";T  
88 END  
  
READY
```

By typing the LIST or LISTNH commands without indicating any line numbers, you can print on the terminal a listing of your entire program. Terminate the command with a carriage return.

```
LISTNH(RET)  
5 FOR I=1 TO 10  
20 INPUT J  
25 LET T=T+J  
50 NEXT I  
55 PRINT "THE TOTAL IS";T  
88 END  
  
READY
```

SCR

Finally, to erase the entire program, which you must do before typing a new program, use the SCR (SCRATCH) command.

```
SCR(RET)  
  
READY
```

All program lines are erased from memory.

**SUMMARY:  
BASIC-11 EDITING  
COMMANDS**

- line #  
Erase the indicated program lines.
- DEL line #  
Erase the indicated program lines.
- LIST  
List the entire program and print a header that includes the program name, date, and time.
- LIST line #  
List the indicated lines and print a header that includes the program name, date, and time.

**LISTNH**

List the entire program but do not print a header.

**LISTNH line #**

List the indicated lines but do not print a header.

**SCR**

Erase all program lines from memory and change the name to NONAME.

**SUB line #@FIRST@SECOND@n**

Replace the nth occurrence of the FIRST character(s) with the SECOND character(s) in the indicated line (default is n=1).

Create the following demonstration program<sup>1</sup>, using the appropriate BASIC-11 editing commands, exactly as it appears here. If you forget to insert a line, type it at the end or when you notice the omission; BASIC-11 sorts and arranges lines by number before execution, regardless of the order in which they are typed. When you have finished, list the entire program and make a final check for typing errors.

```

100 REM THE PROGRAM 23 MATCHES
101 REM
110 PRINT "WE BEGIN WITH 23 MATCHES, YOU MOVE FIRST, YOU"
115 PRINT "MAY TAKE 1, 2, OR 3 MATCHES, TYPE YOUR CHOICE"
120 PRINT "FOLLOWED BY A CARRIAGE RETURN, THEN THE COM-"
125 PRINT "PUTER CHOOSES 1, 2, OR 3 MATCHES, YOU CHOOSE"
130 PRINT "AGAIN, AND SO ON, WHOEVER MUST TAKE THE LAST"
135 PRINT "MATCH, LOSES."
140 PRINT \ LET M=23
200 REM THE HUMAN MOVES
201 REM
210 PRINT \ PRINT "THERE ARE NOW";M;"MATCHES."
215 PRINT \ PRINT "HOW MANY DO YOU TAKE";
230 INPUT H
240 IF H>M THEN 510
250 IF H<>INT(H) THEN 510
260 IF H<=0 THEN 510
270 IF H>=4 THEN 510
280 LET M=M-H
290 IF M=0 THEN 410
300 REM THE COMPUTER MOVES
301 REM
305 IF M=1 THEN 440
310 LET R=M-4*INT(M/4)
320 IF R<>1 THEN 350
330 LET C=INT(3*RND)+1 \ GO TO 360
350 LET C=(R+3)-4*INT((R+3)/4)
360 LET M=M-C
370 IF M=0 THEN 440
380 PRINT \ PRINT "THE COMPUTER TOOK";C;"...";
390 GO TO 310
400 REM SOMEBODY WON
401 REM
410 PRINT \ PRINT "THE COMPUTER WON," \ GO TO 999
440 PRINT \ PRINT "YOU WON," \ GO TO 999
500 REM BAD INPUT
501 REM
510 PRINT "ENTER ONLY 1, 2, OR 3," \ GO TO 215
999 END

```

<sup>1</sup>23 Matches, 101 BASIC Computer Games, Maynard, Mass.: Digital Equipment Corporation, 1975.

As you can see from the first few lines of the listing, this program is a mathematical game where you match your logic against the program logic. The PRINT statements in the program print messages, game instructions, results, and so forth, on the terminal. The REM statements identify comment lines — remarks that provide general information about the program, but that are ignored by BASIC-11 during processing. The INPUT statement in line 230 allows you to supply data from the terminal. Depending on the value you enter, program control transfers to various other parts of the program. For example, if you type an invalid value, program control skips ahead to a PRINT statement in line 510 informing you of your mistake and then returns to line 215 to ask for a value again. The mathematical algorithms of this program are in lines 310 through 350, which determine the number of matches the computer will select based on your choice.

## **RUNNING A BASIC-11 PROGRAM**

**RUN**

Once you have typed the program and checked the listing to be sure that it corresponds to the example, you are ready to run it. The BASIC-11 RUN command initiates program execution. This command prints a header that includes the program name, date, and time. If you want to omit the header line, type the RUNNH command instead.

RUNNH(RET)

If you typed the program correctly, you will see this text print on your terminal:

```
WE BEGIN WITH 23 MATCHES. YOU MOVE FIRST. YOU  
MAY TAKE 1, 2, OR 3 MATCHES. TYPE YOUR CHOICE  
FOLLOWED BY A CARRIAGE RETURN. THEN THE COM-  
PUTER CHOOSES 1, 2, OR 3 MATCHES. YOU CHOOSE  
AGAIN, AND SO ON. WHOEVER MUST TAKE THE LAST  
MATCH, LOSES.
```

```
THERE ARE NOW 23 MATCHES.
```

```
HOW MANY DO YOU TAKE?
```

### **NOTE**

If this response does not appear, you have not entered the program correctly. Compare your listing very carefully against the one provided earlier. Spacing does not matter, but all other characters must match. To correct your errors type CTRL/C, which, under control of BASIC-11 only, returns you to BASIC-11 command mode, indicated by the READY message. Correct the program and then rerun it.

When the program pauses and asks you a question, you must supply data, in this case a 1, 2, or 3. Type your choice (represented here by  $n$ ), followed by a carriage return:

```
n␣
```

```
?SYNTAX ERROR AT LINE 250
```

```
READY
```

BASIC-11 discovered an error<sup>1</sup> in line 250 that prevents further processing. Check line 250 in your listing or list it on the terminal:

```
LISTNH 250␣
```

```
250 IF H<>INT(H THEN 510
```

```
READY
```

Note that a right parenthesis is missing after the second H in this line. Correct the line using the SUBSTITUTE command:

```
SUB 250 @ (H@ (H) @␣
```

```
250 IF H<>INT(H) THEN 510
```

```
READY
```

You are ready to run the program again.

```
RUNNH␣
```

BASIC-11 begins processing at the start of the program.

```
WE BEGIN WITH 23 MATCHES. YOU MOVE FIRST. YOU
MAY TAKE 1, 2, OR 3 MATCHES. TYPE YOUR CHOICE
FOLLOWED BY A CARRIAGE RETURN. THEN THE COM-
PUTER CHOOSES 1, 2, OR 3 MATCHES. YOU CHOOSE
AGAIN, AND SO ON. WHOEVER MUST TAKE THE LAST
MATCH, LOSES.
```

```
THERE ARE NOW 23 MATCHES.
```

```
HOW MANY DO YOU TAKE?
```

Type your choice again. But notice this time that a different kind of error is detected. The BASIC-11 interpreter has entered an infinite loop, a series of commands that it repeats endlessly. After several lines have printed, type a double CTRL/C; this interrupts execution and returns control to BASIC-11 command mode.

CTRL/C CTRL/C

<sup>1</sup>Refer to the *RT-11 System Message Manual* for greater detail about any messages printed during normal system use.

```
n(RET)
THE COMPUTER TOOK 1 ....
THE COMPUTER TOOK 1 ....
THE COMPUTER TOOK 3 ....
THE COMPUTER TOOK 2 ....
THE COMPUTER TOOK 2 ....
THE COMPUTER TOOK 3 ....
THE COMPUTER TOOK 3 ....
THE COMPUTER TOOK 1 ....
THE COMPUTER TOOK 1 ....
THE COMPUTER TOOK 3 ....
THE COMPUTER TOOK 1 ....
THE COMPUTER TOOK 3 ....
(CTRL/C) (CTRL/C)
STOP AT LINE 380

READY
```

An infinite loop is a programming logic error. However, since the error does not prevent processing, BASIC-11 does not print an error message. Instead BASIC-11 is caught in a loop of instructions and executes them endlessly. This particular loop is obvious because it prints a line of text; other kinds of loops may not be so evident. At this point you must examine the program logic to determine why these instructions are being repeated.

Look at your listing of this program. The problem in this case is in line 390. This line instructs program control to return to line 310; therefore lines 310 through 390 are repeated endlessly without ever obtaining your next value choice. Program control should really return to line 210. Correct line 390 as follows:

```
SUB 390 @310@210@(RET)
390 GO TO 210

READY
```

Now you are ready to run the program again. This time the entire program should execute without error. Enter your value choices when requested. (A hint to playing the game: your first value choice determines whether you can win; if your first choice is wrong, the program has the advantage throughout.) A sample run follows.

```
RUNNH(RET)

WE BEGIN WITH 23 MATCHES. YOU MOVE FIRST. YOU
MAY TAKE 1, 2, OR 3 MATCHES. TYPE YOUR CHOICE
FOLLOWED BY A CARRIAGE RETURN. THEN THE COM-
PUTER CHOOSES 1, 2, OR 3 MATCHES. YOU CHOOSE
AGAIN, AND SO ON. WHOEVER MUST TAKE THE LAST
MATCH, LOSES.

THERE ARE NOW 23 MATCHES.
```

HOW MANY DO YOU TAKE? 1<sup>(RET)</sup>

THE COMPUTER TOOK 1 ....  
THERE ARE NOW 21 MATCHES.

HOW MANY DO YOU TAKE? 1<sup>(RET)</sup>

THE COMPUTER TOOK 3 ....  
THERE ARE NOW 17 MATCHES.

HOW MANY DO YOU TAKE? 2<sup>(RET)</sup>

THE COMPUTER TOOK 2 ....  
THERE ARE NOW 13 MATCHES.

HOW MANY DO YOU TAKE? 1<sup>(RET)</sup>

THE COMPUTER TOOK 3 ....  
THERE ARE NOW 9 MATCHES.

HOW MANY DO YOU TAKE? 1<sup>(RET)</sup>

THE COMPUTER TOOK 3 ....  
THERE ARE NOW 5 MATCHES.

HOW MANY DO YOU TAKE? 3<sup>(RET)</sup>

THE COMPUTER TOOK 1 ....  
THERE ARE NOW 1 MATCHES.

HOW MANY DO YOU TAKE? 0<sup>(RET)</sup>  
ENTER ONLY 1, 2, OR 3.

HOW MANY DO YOU TAKE? 1<sup>(RET)</sup>

THE COMPUTER WON.

READY

#### RUN

Execute the BASIC-11 program currently in memory; print a header line including the program name, date, and version number.

#### RUNNH

Execute the BASIC-11 program currently in memory; omit the header line.

#### CTRL/C

Under control of BASIC-11 only, interrupt execution of the BASIC-11 program and return control to BASIC-11 command mode.

#### BYE

Return control to monitor command mode (only when using BASIC-11).

**SUMMARY:  
BASIC-11  
EXECUTION  
COMMANDS**

## FILE MAINTENANCE

You can transfer the BASIC-11 program currently in memory to a storage volume by using the SAVE command of BASIC. The SAVE command copies the program to the storage volume and gives the program the file name and file type that you indicate in the command line. A file type of .BAS is assigned automatically unless you indicate otherwise.

SAVE

Use the SAVE command to store this BASIC-11 program as MATCH.BAS on the storage volume (VOL:) as follows:

```
SAVE VOL:MATCHRET
```

```
READY
```

NEW

After you save a BASIC-11 program on a storage volume, you can create a new program in memory by typing the BASIC-11 NEW command. This command erases the current memory contents and asks you for a new program name:

```
NEWRET  
NEW FILE NAME--
```

You can type any file name you wish and BASIC-11 assigns it to the file you create. Or you can respond by typing only a carriage return; BASIC-11 then assigns the file name NO-NAME.

Another way to create a new program in memory is to type the BASIC-11 SCR command. This command simply erases the current memory contents. It assigns the name NONAME:

```
SCRRET
```

```
READY
```

OLD

To use an existing BASIC-11 program, one that you have previously stored on a storage volume, type the BASIC-11 OLD command:

```
OLDRET  
OLD FILE NAME--
```

Reply by typing the device name, file name, and file type of the file that you want to use. If you omit an explicit device name, BASIC-11 assumes DK: (the default volume); if you omit an explicit file type, BASIC-11 assumes .BAS. BASIC-11 erases memory and then copies the program from the volume into memory. For example, type:

```
VOL:MATCHRET
```

```
READY
```

This copies VOL:MATCH.BAS back into memory.

Assume that you have edited or changed the MATCH.BAS file and now want to transfer it back to VOL:. Since the file already

exists as MATCH.BAS on that volume, you must use the BASIC-11 REPLACE command:

```
REPLACE VOL: MATCH(RET)
```

```
READY
```

The REPLACE command replaces an existing file with a new version.

REPLACE

The SAVE and REPLACE commands copy a BASIC-11 program from computer memory to a storage volume. As these commands copy the program, they convert it from the internal format used by BASIC-11 to ASCII format. Thus, you can, if you prefer, use the RT-11 editor to create and edit BASIC-11 programs, since the editor also uses ASCII format. However, many users would rather use BASIC-11 to create and edit a BASIC-11 program, since they can then run the program, reedit it, rerun it, and save the new version — all in BASIC-11 command mode — rather than perform the several corresponding monitor commands.

The last file maintenance operation that you should perform is to obtain an up-to-date directory of your storage volume so that you can see its current status; however, you must return to monitor command mode to do this. Type the BYE command; this BASIC-11 command (rather than CTRL/C) returns control to monitor command mode. Next use the DIRECTORY monitor command to check the status of your storage volume.

```
BYE(RET)
```

```
.DIRECTORY/BRIEF VOL:
 08-Jan-83
SUM .MAC EXAMP .FOR EXAMP .MAC GRAPH .FOR GRAPH .LST
GRAPH .OBJ GRAPH .SAV MATCH .BAS
 8 Files, 58 Blocks
4704 Free blocks
```

## NEW

Create a new BASIC-11 program, assigning the file name indicated.

## OLD

Copy into memory an existing BASIC-11 program (for use under BASIC-11).

## REPLACE

Copy the BASIC-11 program currently in memory to the indicated storage volume, replacing the version that already exists on that volume.

## SAVE

Copy the BASIC-11 program currently in memory to the indicated storage volume.

**SUMMARY:  
BASIC-11 FILE  
MAINTENANCE  
COMMANDS**

This completes the BASIC-11 demonstration. Before you continue to Chapter 14 to learn about program debugging, make sure that the main system volume is loaded in device unit 0. If you followed the special instructions in Appendix B to load the language volume, you should now stop the system, unload that volume, load the main system volume, and rebootstrap the system.

## **REFERENCES**

*BASIC-11 Language Reference Manual* (AA-1908A-TC). Maynard, Mass.: Digital Equipment Corporation, 1976.

A reference manual and guide to programming in the BASIC-11 language.

*BASIC-11/RT-11 Installation Guide and Release Notes* (AA-K724B-TC). Maynard, Mass.: Digital Equipment Corporation, 1983.

An RT-11-specific manual that contains instructions for installing the RT-11 BASIC-11 language processor and lists known problems and differences between versions.

*BASIC-11/RT-11 User's Guide* (AA-5071B-TC). Maynard, Mass.: Digital Equipment Corporation, 1983.

An RT-11-specific manual that contains information necessary to create, edit, run, and debug a BASIC-11 program.

# CHAPTER 11

## RUNNING A MACRO-11 ASSEMBLY LANGUAGE PROGRAM

The MACRO-11 programming language is a machine-dependent programming language developed for the PDP-11 programmer, or for the FORTRAN IV programmer who intends to combine assembly language routines and FORTRAN IV routines. The MACRO-11 language enables the knowledgeable programmer to access all the features of the RT-11 computer system using a precise and efficient programming code.

The MACRO-11 assembly language uses the PDP-11 instruction set, a list of mnemonic instructions that correspond to various PDP-11 computer operations. These instructions allow you to add, compare, increment, complement, and perform many other manipulations on numerical data. The instructions are summarized in a pocket-sized folding card, called the *PDP-11 Programming Card* (Figure 11-1), and are described in detail in the *PDP-11 Processor Handbook*. By choosing the appropriate instructions and by providing any additional data needed, you can create a complete program.

### DEVELOPING A MACRO-11 ASSEMBLY LANGUAGE PROGRAM

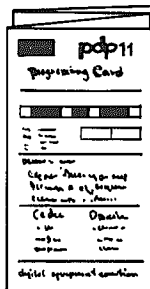


Figure 11-1 PDP-11 Programming Card

You write the MACRO-11 program as a sequence of lines, each a single assembly language statement in the following format:

**LABEL: OPERATOR OPERAND(S) COMMENTS**

The operator and/or operand are instructions selected from the PDP-11 instruction set, data needed by the instructions, or assembler directives (instructions to the assembler to guide the assembly process). The optional statement label identifies the statement line so that you can refer to the instructions or data on that line from other parts of the program. Optional comments describe generally what operations are being done. Sequences of language statements constitute a routine (to perform a specific function); groups of routines and data compose the entire executable program.

When you are satisfied with the logic of your MACRO-11 source program, you use the RT-11 editor to create it as a file (see Chapter 5). You use tabs and spaces to make the program more readable. When you have finished creating the program as a complete, edited file, you next enter it as input to the MACRO-11 language processor, which is part of the RT-11 operating system and is stored on your system volume. The MACRO-11 language processor processes (assembles) the language statements, converting them into an internal machine language code called object code. This code is next processed by the system linker, which combines your program units and makes the program suitable for execution. Figure 11-2 represents the development of an executable MACRO-11 program.

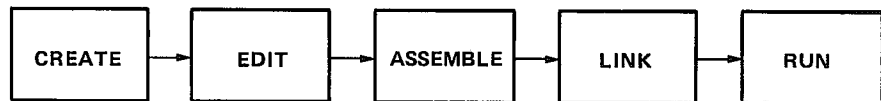


Figure 11-2 Evolution of a MACRO-11 Program

### USING THE MACRO-11 LANGUAGE PROCESSOR

The MACRO-11 language processor is an assembler that accepts information in one format (that is, your source program) and translates it into another format (that is, a machine language program). The assembler interprets and processes the assembly language statements, one at a time, and generates one or more computer instructions or data items. Since you originally use the editor to create a MACRO-11 program in ASCII format, you must next translate it into a machine format that the computer can use. The MACRO-11 assembler performs this conversion, producing as output a new version of the program, in object format, called an object module. You may request the MACRO assembler to produce a listing of the source program at the same time. The role of the assembler is represented below in Figure 11-3.

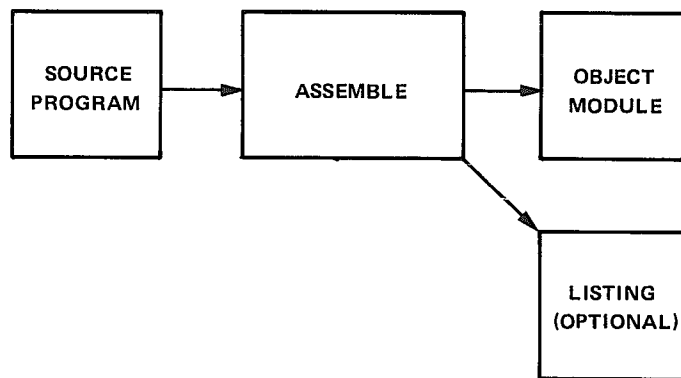


Figure 11-3 Function of a MACRO-11 Assembler

During assembly processing, the MACRO-11 assembler:

- Accounts for all instructions used within the source program and determines their relative positions in computer memory; it does this by means of a storage location (program) counter.
- Keeps track of all user-defined symbols and their respective values in a symbol table.
- Converts assembly language mnemonics, user-defined symbols, and data values into their respective machine language (object code) equivalents.

The program counter keeps track of addresses in computer memory where instructions and data will be stored.

### The Program Counter

PDP-11 computer memories are composed of physical storage locations that can hold numerical data. These locations are numbered consecutively from 0 up to the highest memory location, which varies according to the amount of memory acquired with the computer system (Figure 11-4). PDP-11 computers used in an RT-11 system have at least 32,768 bytes (16,384 words).

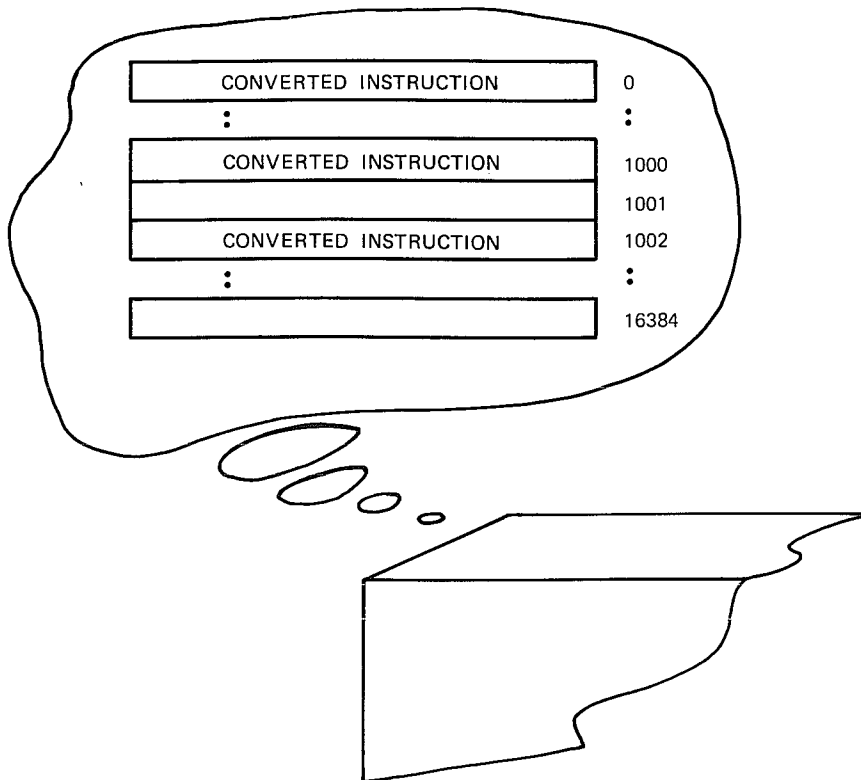


Figure 11-4 PDP-11 Computer Memory

During processing, the assembler converts each program language statement into numerical data (the object code) and assigns the data a relative storage location. The system linker will convert the relative storage locations assigned by the assembler to absolute storage locations in the computer memory.<sup>1</sup> The location's associated number is called its address. As the assembler translates and assigns each statement, it updates the value of the program counter accordingly.

## The Symbol Table

Since you may not know which locations, or how many locations, the program needs, you use symbolic names (variables, constants, and labels) to represent individual locations and their contents. As the assembler processes the source program, it constructs a symbol table, which is a compiled list of all the symbolic names and labels that you have used within the program. The MACRO-11 assembler defines each symbolic name by assigning an address or data value, as appropriate, and adds the symbol definition to the symbol table. After assembly, you can refer to the symbol table, which is printed at the end of the assembly listing, to find all symbol definitions.

## Machine Language Code

The third function of the assembler is to convert your MACRO-11 source language statements into machine language code (the object module).

### NOTE

The following information will help you understand the assembly listing used later in this chapter.

Machine language code is numerical data in the form of binary numbers (numbers composed of only the digits 0 and 1). Binary numbers are appropriate because the digits 0 and 1 can be easily manipulated by the two-state electronic logic of the computer.

For example, a typical assembled instruction in PDP-11 computer memory looks like this:

location address	location contents
.	.
.	.
01000	11000000
01001	11100101
.	.
.	.

<sup>1</sup>The system linker is discussed in Chapter 12.

Since a single instruction requires two (or more) consecutive memory locations, the instruction is actually put together in memory in the following manner:

```

high-order byte  low-order byte
01001 1 1 1 0 0 1 0 1  1 1 0 0 0 0 0 0  01000
    
```

Each individual digit of the instruction is called a bit (binary digit). A single memory location, called a byte, contains 8 bits; two memory locations, called a PDP-11 word, contain 16 bits.

The byte in the even-numbered memory address is called the low-order byte and is stored first; the byte in the odd-numbered memory address is called the high-order byte and is stored next. Both bytes together form one PDP-11 16-bit word (Figure 11-5).

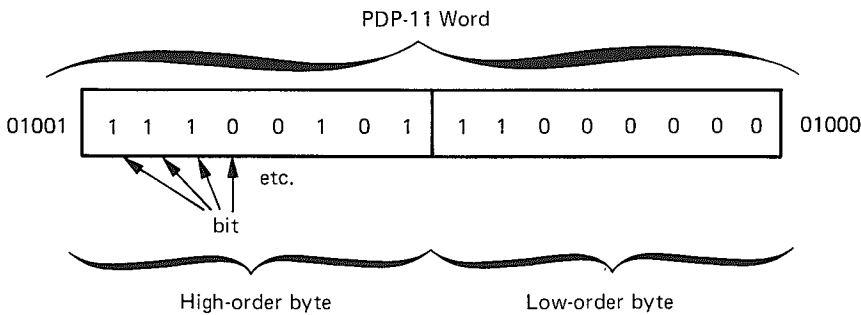


Figure 11-5 PDP-11 Word

The computer works in terms of 8-bit bytes and 16-bit words of binary data. However, binary numbers are generally too long and cumbersome to be used effectively by a programmer. But binary numbers can be easily represented as octal numbers (numbers composed of digits within the range 0 to 7). Since octal numbers are closer to the familiar decimal number system and are much more readable than binary numbers, the programmer more often uses octal numbers than binary numbers.

Table 11-1 shows the decimal numbers 0 through 10 and their respective octal and binary equivalents. Tables and formulas are available to calculate higher conversions.

Thus, you can think of the binary instruction shown earlier in terms of its octal equivalent as follows (conversion is done from low-order to high-order byte in groups of three bits):

```

high-order byte  low-order byte
01001 1 1 1 0 0 1 0 1  1 1 0 0 0 0 0 0  01000

  1  6  2      7  0  0  = 162700(8)
    
```

A MACRO-11 assembly listing shows the addresses of memory locations and their contents as octal numbers. The octal numbers represent the respective binary machine language code that makes up the object module.

**Table 11-1 Decimal/Octal/Binary Conversion**

Decimal	Octal	Binary
0	0	000
1	1	001
2	2	010
3	3	011
4	4	100
5	5	101
6	6	110
7	7	111
8	10	1 000
9	11	1 001
10	12	1 010

## ASSEMBLING THE MACRO-11 PROGRAM

In Chapter 5 you used the RT-11 editor to create a MACRO-11 source program; you then stored it on your storage volume. Since a source program is in ASCII format, the next step is to use the MACRO-11 assembler to convert the source program to object code.

Copy the MACRO-11 source program from the storage volume back to the system volume (which is the default volume for input/output operations).

On your storage volume are two MACRO-11 source programs, the one you created, SUM.MAC, and the one provided for you, DEMOX1.MAC, which was renamed to EXAMP.MAC in Chapter 7. Which of these you should copy depends on the results of the source comparison you performed in Chapter 6. If the comparison resulted in no differences except for the title lines, copy your own program (SUM.MAC) as follows:

### Long Command Format

```
.COPY␣  
From? VOL:SUM.MAC␣  
To ? SUM.MAC␣
```

### Short Command Format

```
.COPY VOL:SUM.MAC SUM.MAC␣
```

However, if differences were listed in addition to the title lines, substitute the program EXAMP.MAC:

**Long Command Format**

```
.COPY(RET)
From? VOL:EXAMP,MAC(RET)
To ? SUM,MAC(RET)
```

**Short Command Format**

```
.COPY VOL:EXAMP,MAC SUM,MAC(RET)
```

Whichever source file you copied now resides on your system volume under the name SUM.MAC and is the file that you will process with the MACRO-11 assembler. The command used to assemble a MACRO-11 source program is the monitor MACRO command.

Use the MACRO command with its /LIST and /CROSSREFERENCE options to assemble your source program and produce a cross-referenced assembly listing. The system prompt asks you to supply the input file name. You can omit typing the .MAC file type, since the MACRO command assumes this file type unless you indicate otherwise. The system will automatically assign the name SUM.OBJ to the object module and SUM.LST to the listing file, and store both newly created files on the system volume. (The system volume is the default storage volume for input/output operations.)

MACRO

**Long Command Format**

```
.MACRO(RET)
Files? SUM/LIST/CROSSREFERENCE(RET)
```

**Short Command Format**

```
.MACRO SUM/LIST/CROSSREFERENCE(RET)
```

Assembly begins. When it is finished, a message similar to the following prints on the terminal printer or screen:

```
?MACRO-W-Errors detected: 6
DK:SUM,DK:SUM/C=DK:SUM
```

This message indicates that the assembler detected errors in six lines of the source program during processing. It helps at this point to look at the listing produced by the assembler for information.

**Long Command Format**

(Line Printer)

```
.PRINT(RET)
Files? SUM,LST(RET)
```

(Terminal)

```
.TYPE(RET)
Files? SUM,LST(RET)
```

Short Command Format

(Line Printer)

(Terminal)

.PRINT SUM,LST<sup>RET</sup>

.TYPE SUM,LST<sup>RET</sup>

Your listing should look like the following example. An explanation of this listing follows. You should refer to the listing as you read the accompanying explanation.

NOTE

You do not need to understand the MACRO-11 language or the way this program works to successfully complete the exercises in this chapter.

```

SUM.MAC VERSION 1      MACRO V05.00 Saturday 08-Jan-83 09:21 Page 1

1                          .TITLE SUM.MAC VERSION 1
2
3                          .MCALL .TTYOUT, .EXIT, .PRINT
4
5
6
7          000106          ; N = 70.          ;NO. OF DIGITS OF 'E' TO CALCULATE
8          ;              ; 'E' = THE SUM OF THE RECIPROCAL OF THE FACTORIALS
9          ;              ; 1/0! + 1/1! + 1/2! + 1/3! + 1/4! + 1/5! + ...
10
M 11 000000          EXP: .PRINT #MESSAG          ;PRINT INTRODUCTORY TEXT
12 000006 012705 000106          MOV #N,R5          ;NO. OF CHARS OF 'E' TO PRINT
13 000012 012700 000107          FIRST: MOV #N+1,R0          ;NO. OF DIGITS OF ACCURACY
U 14 000016 012701 000000          MOV #A,R1          ;ADDRESS OF DIGIT VECTOR
15 000022 006311          SECOND: ASL R1          ;DO MULTIPLY BY 10 (DECIMAL)
16 000024 011146          MOV R0,R1,-(SP)          ;SAVE #2
17 000026 006311          ASL R1          ;#4
18 000030 006311          ASL R1          ;#8
19 000032 062621          ADD (SP)+,(R1)+          ;NOW #10, POINT TO NEXT DIGIT
20 000034 005300          DEC R0          ;AT END OF DIGITS?
21 000036 001371          BNE SECOND          ;BRANCH IF NOT
22 000040 012700 000106          MOV #N,R0          ;GO THRU ALL PLACES, DIVIDING
23 000044 014103          THIRD: MOV -(R1),R3          ;BY THE PLACES INDEX
24 000046 012702 177777          MOV #-1,R2          ;INIT QUOTIENT REGISTER
25 000052 005202          FOURTH: INC R2          ;BUMP QUOTIENT
26
27 000054 160003          SUB R0,R3          ;SUBTRACT LOOP ISN'T BAD
28 000056 103375          BCC FOURTH          ;NUMERATOR IS ALWAYS < 10*N
29 000060 060003          ADD R0,R3          ;FIX REMAINDER
30 000062 010311          MOV R3,R1          ;SAVE REMAINDER AS BASIS
31
AR 32 000064 066167 000000 000000          ADD R2-2,(R1)          ;GREATEST INTEGER CARRIES
33          ;TO GIVE DIGIT
34 000072 005300          DEC R0          ;AT END OF DIGIT VECTOR?
35 000074 001363          BNE THIRD          ;BRANCH IF NOT
36 000076 014100          MOV -(R1),R0          ;GET DIGIT TO OUTPUT
37 000100 162700 000012          FIFTH: SUB #10,,R0          ;FIX THE 2.7 TO .7 SD
38          ;THAT IT IS ONLY 1 DIGIT
39 000104 103375          BCC FIFTH          ;(REALLY DIVIDE BY 10)
U 40 000106 062700 000072          ADD #10,+0,R0          ;MAKE DIGIT ASCII
41 000112 000000          .TTYON          ;OUTPUT THE DIGIT
42 000114 005011          CLR R1          ;CLEAR NEXT DIGIT LOCATION
43 000116 005305          DEC R5          ;MORE DIGITS TO PRINT?
44 000120 001334          BNE FIRST          ;BRANCH IF YES
45 000122          .EXIT          ;WE ARE DONE
46
H 47 000124 000107          EXP: .REPT N+1          ;INIT VECTOR TO ALL ONES
48          .WORD 1
49          .ENDR
50
51 000342 124 110 105 MESSAG: .ASCII /THE VALUE OF E IS: /<15><12> /2./ <200>
52          000345 040 124 101
53          000350 114 125 105
54          000353 040 117 106
55          000356 040 105 040
56          000361 111 123 072
57          000364 015 012 062

```

```

SUM.MAC VERSION 1      MACRO V05.00 Saturday 08-Jan-83 09:21 Page 1-1

52          000347 056 200          .EVEN
53
D 54          000000'          .END EXP

```

```

SUM.MAC VERSION 1      MACRO V05.00 Saturday 08-Jan-83 09:21 Page 1-2
Symbol table
A - *****          FIFTH 000100R          FOURTH 000052R          N = 000106          THIRD 000044R
EXP 000000R          FIRST 000012R          MESSAG 000342R          SECOND 000022R          .TTYON= *****

```

```

. ABS. 000000 000 (R0,I;GBL;ABS;OVR)
000372 001 (R0,I;LCL;REL;CDN)
Errors detected: 6

*** Assembler statistics
Work file reads: 0
Work file writes: 0
Size of work file: 8222 Words ( 33 Pages)
Size of core pool: 15872 Words ( 62 Pages)
Operating system: RT-11

```

Elapsed time: 00:00:04.34  
 DK:SUM,DK:SUM/C=DK:SUM

SUM.MAC VERSION 1 MACRO V05.00 Saturday 08-Jan-83 09:21 Page S-1  
 Cross reference table (CREF V05.00)

```
.TTYON 1-41
A 1-34
EXP 1-11# 1-47# 1-54
FIFTH 1-37# 1-39
FIRST 1-13# 1-44
FOURTH 1-25# 1-28
MESSAG 1-11 1-51#
N 1-7# 1-12 1-13 1-22 1-47
SECOND 1-15# 1-21
THIRD 1-23# 1-35
```

SUM.MAC VERSION 1 MACRO V05.00 Saturday 08-Jan-83 09:21 Page M-1  
 Cross reference table (CREF V05.00)

```
.EXIT 1-3# 1-45
.PRINT 1-3# 1-11
.TTYOU 1-3#
```

SUM.MAC VERSION 1 MACRO V05.00 Saturday 08-Jan-83 09:21 Page E-1  
 Cross reference table (CREF V05.00)

```
A 1-32
D 1-54
M 1-11 1-47
R 1-32
U 1-14 1-41
```

The first part of the listing has four logical sections, as follows:

line number	octal memory address	octal instruction value(s)	statement line
----------------	----------------------------	----------------------------------	----------------

The assembler assigns consecutive decimal line numbers to each line of the source program, including blank lines and comment lines. These numbers are used for reference purposes. The next column to the right shows the relative<sup>1</sup> even-numbered octal memory (byte) addresses of storage locations assigned by the program counter to each instruction in the program. This program has been assigned relative memory addresses 0 through 370. The third column (and possibly fourth and fifth) shows the octal equivalent of the assembled instruction or data value. An apostrophe following an octal value indicates a relative value that must be modified before it can be used (the actual value is determined during linking). Finally, the source program as you created it appears in the right-hand portion of the listing.

For example, look at line 18 of the listing:

```
18 000030 006311 ASL @R1 ;*8
```

The instruction ASL @R1 is stored in relative memory locations 30 and 31 as binary data (the comment, ;\*8, is ignored):

```
31 0 0 0 0 1 1 0 0 1 1 0 0 1 0 0 1 30
    0 0 6 3 1 1
```

<sup>1</sup>The assembler assigns relative memory addresses to instructions. Actual addresses are not determined until the link operation is performed. Linking and address relocation are discussed in Chapter 12.

Some instructions require more than two memory locations; for example, those at lines 13 and 14. The number of memory locations required depends upon the operation.

Following the assembled code in the listing is the symbol table, an alphabetical listing of user-defined symbols and labels in the program and their respective definitions. Symbols are defined as values. For example, the symbolic variable name N is defined (in line 7) as 000106(octal) or 70(decimal), an absolute value. Labels are defined as addresses. The symbolic label FIRST is defined (in line 14) as 000012, a relocatable address (the R following 000012 in the symbol table indicates that the address will be relocated or modified during linking). A row of asterisks next to any symbolic name in the table indicates that for some reason — possibly a programming error — the assembler could not define the symbol.

At the very end of the symbol table (where the . ABS. occurs) is the program's size information (or synopsis) in terms of the total number of octal storage locations it requires (in this case, 372). Following is the number of errors detected, and the amount of free and used memory pages (statistics provided by the assembler).

Following the symbol table is the cross reference (CREF) listing. The CREF listing is optional — as is the assembly listing — but provides you with useful reference and debugging information, especially if the program is large. The CREF listing can contain several kinds of tables of reference information, each beginning on a new page. The default tables are the three shown here.

Every reference in a CREF table shows the page number of the listing (in the preceding example, all references are on page 1), followed by the appropriate line number. A number sign following a line number indicates that this line is where a label or symbol definition occurs.

The first CREF table shown here lists alphabetically all user-defined symbol and label references.

The second CREF table lists alphabetically all macro symbol references. (Macro symbols are a special feature of the MACRO-11 assembly language; they are described shortly.)

The third CREF table lists alphabetically the codes of the errors detected during assembly. These errors must be corrected before you can run the program.

Now that you are familiar with the format of an assembly listing, go back to the beginning of the example listing to determine what this program should do.

The first two comment lines (preceded by semicolons) indicate that the program calculates the value of 'E', which is the sum of the inverse of the factorials between 1 and infinity. The algorithm used in this program is somewhat complicated (this was necessary to keep the program reasonably short). 'E' is calculated one digit at a time by using a difference function between its actual value and the current approximation for each new digit. The program forms:

$$1 + (1 + (1 + \dots + (1 + ((1 + (1/N))/(N-1))/(N-2))/\dots/2)/1)$$

and is 2.11111... in the inverse factorial base system, which is the first sum shown in the program listing.

The statements between lines 1 through 7 define initial states to the assemblers, such as the value of N, and designate the macros that will be used throughout the program.

Macros, from which the MACRO-11 language processor derives its name, are a useful feature of the MACRO-11 assembly language. You can define as a macro any recurring sequence of coding instructions. By giving the macro a name, you can thereafter call it from any other part of the program by using a single language statement.

In addition to the macros you define yourself, the RT-11 system provides system macros that your programs can access. System macros are defined in a special system library file called SYS-MAC.SML (SML stands for System Macro Library). SYS-MAC.SML is part of the RT-11 operating system and is stored on the system volume. If you request a system macro from your source program, the MACRO-11 assembler automatically searches SYSMAC.SML for the required information.

The system macros defined in SYSMAC.SML are calls to certain services performed by the RT-11 monitor, such as terminal handling, input and output operations, program termination, file capabilities, and so on. The portion of the monitor that performs these services, or that is capable of getting the necessary program code to perform these services, is always in memory and is therefore called the resident monitor. Thus, whenever your source program is in memory and is to be executed, the resident monitor is also there with its services.

You communicate the need for a monitor service by issuing a programmed request in your source program. A programmed request consists simply of a macro call to a specific macro defined in SYSMAC.SML. The macro expands into the appropriate machine language code, which, during program execution, makes a request to the resident monitor to supply the desired service.

You specify all programmed requests that you intend to use in your source program in an `.MCALL` statement, like the one shown at line 3 in the listing. For example, the programmed request `.TTYOUT` requests the monitor to print an ASCII character on the console terminal. During assembly, the `.TTYOUT` macro in `SYSMAC.SML` is expanded into machine language code. During program execution this code requests the resident monitor to take the indicated ASCII character and send it to the console terminal.

Line 12 in the program uses another programmed request, `.PRINT`, to print a message on the terminal.

Lines 13 through 15 are initialization instructions: they set initial values in three of the special registers. Lines 16 through 22 represent a routine that does a multiplication by 10. Lines 23 and 24 are setup instructions for the division routine of lines 25 through 28. Lines 29 through 35 save the quotient and remainder. Lines 36 through 40 print the digits of *E*. Lines 43 and 44 count the number of digits.

The statements at lines 47 through 49 reserve a buffer area (a series of locations in memory) to be used by the program and therefore not to be assigned to other instructions. The statement at line 51 provides the data for printing the ASCII text message *THE VALUE OF E IS: 2*.

This program, however, contains errors. The assembler discovered six lines with errors that prevent the program from assembling properly. The assembler flags (points out) errors by printing a code letter in the assembly listing or on the terminal if no listing is requested.<sup>1</sup>

The first error occurs at line 12 and is an M error. This means a label was defined more than once. You can refer to a label any number of times, but you may define it only once. By looking at the CREF user symbol table, you can see that the label is defined at line 12 and again at line 47; one of these definitions is wrong. Examination of the program logic reveals that the definition at line 12 is correct. Before deciding how to change line 47, though, check the other errors to see if one of them indicates what should be done. In fact, the next error encountered (line 15) shows what is wrong. A U error identifies an undefined symbol. The label *A* is referenced in line 15, but is never defined within the program. It should be defined logically at line 47. Therefore, line 47 should be changed to read:

```
A:      .REPT   N+1
```

---

<sup>1</sup>Refer to the *RT-11 System Message Manual* for greater detail about any system messages printed during normal system use.

Thus, this one change eliminates three errors flagged by the assembler; those at lines 12, 15, and 47.

The next error occurs at line 32. Actually, the assembler flagged two errors here. An A error indicates an addressing problem and an R error indicates a register error (invalid use of a register, a special PDP-11 storage feature). If you look at the language statement in line 32, you can see that the ADD operator is followed by one operand. However, ADD is an instruction that requires two operands (two values to be added together) separated by a comma. This statement simply contains a typing error, which can be corrected by inserting a comma between the R2 and the -2(R1). Thus, changing the line as follows both corrects the addressing problem and eliminates the invalid register expression:

```
ADD      R2,-2(R1)
```

At line 41 is another undefined symbol, the macro symbol .TTYON. Since the program designated the macro symbol .TTYOUT in line 3, this error indicates a misspelling. Correct line 41 to read:

```
.TTYOUT
```

Finally, a D error occurs in line 54. This indicates that reference was made to a symbol that is defined more than once. This error has already been eliminated as a result of the correction made to line 47.

Thus, by changing the three lines indicated, you can correct all the errors flagged during assembly. So the next step is to edit the appropriate lines in the source program. If necessary, review the editing commands in Chapter 5, and then edit the file SUM.MAC on your system volume so that the three lines indicated are error-free. Do not rename the file. When you are ready, reassemble the program, using the MACRO command, and obtain a new object module and a new listing. This time the program should assemble without error. If errors occur, you have not edited the program correctly. Compare listings and try to correct your errors, or go back to the beginning of this chapter and repeat the demonstration.

The object module produced by the MACRO command may in itself be incomplete. It may need to be joined with other object modules or library files to form a complete functioning program,<sup>1</sup> since all required object modules must be joined before the program can work.

## LINKING OBJECT MODULES TOGETHER

<sup>1</sup>Chapters 12 and 13 give more information on linking files and using library files, respectively.

Thus, you must next link the SUM object module with any other object modules it requires. However, the only file used by this program was the macro library file SYSMAC.SML, and it was used during assembly. So in this case, you do not need to join the SUM object module with any other modules.

NOTE

Some other MACRO-11 programs that you write later may reference system subroutines supplied in the system subroutine library, SYSLIB.OBJ. Programs that reference these routines must be linked with the system subroutine library to satisfy external references. If SYSLIB.OBJ is not present on your system volume, follow the guidelines in the section of Appendix B entitled Using the LINK Volume.

Even though SYSLIB is not required for SUM.OBJ, you must still link the file. The link operation, in addition to joining object modules together, also assigns absolute memory addresses to the relative addresses calculated by the MACRO-11 assembler. Since the memory addresses of one object module must be relocated to accommodate addresses used in another object module, the link operation serves to resolve all address changes. The result of the link is a memory image load module, with all module links resolved and all absolute memory addresses and storage information assigned (Figure 11-6). The memory image module, then, is actually a picture of what computer memory looks like just before program execution.

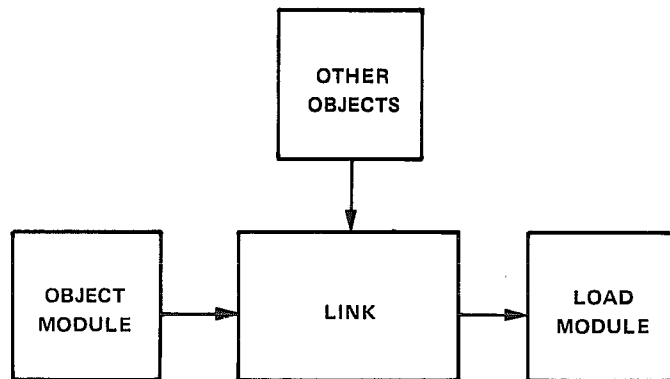


Figure 11-6 The Link Operation

LINK

To link the object modules, use the LINK command. The system prompts you to enter the names of the input object modules to be linked together. You can omit typing the .OBJ file type in the command line since the LINK command assumes this file type for input. After you have entered the input information, the system begins linking the object module. You do not have to

specify an output file, since the system automatically assigns the file name of the first input file and a file type of .SAV to the output file.

### Long Command Format

```
.LINK(RET)
Files? SUM(RET)
```

### Short Command Format

```
.LINK SUM(RET)
```

Any messages printed inform you of error conditions discovered during the link operation (for example, if you fail to specify all the necessary input object modules). However, assuming you edited your source program correctly and that it assembled without error, it should also now link without error.

A load module is one that you can run on the system. Unless your program contains logic errors that prevent it from running properly (errors that the system cannot always detect), running the .SAV version of your program should produce the results you intended. However, if logic errors exist within your program, running the program will produce either erroneous results or none at all. If this is the case, you must study the source program, rework it, reedit it, then perform the assembly and link operations again.

If your MACRO-11 program is error-free, running the .SAV version should produce the expected results. In this demonstration, running the SUM.SAV file should produce a value on the terminal that is the constant E (2 followed by 70 digits).

## RUNNING THE MACRO-11 PROGRAM

To execute the MACRO-11 demonstration program, use the monitor RUN command. You can omit typing the .SAV file type, since the RUN command assumes this file type. Type the following, and note the results printed on the terminal:

### Long and Short Command Format

```
.RUN SUM(RET)
THE VALUE OF E IS:
2.5/606/606237.2301314.06525/130440275535025.71477737352744745405502.544
.
```

You can see that something is wrong. Slashes and periods appear in the result, indicating that an error still exists somewhere in the program.

Programming errors, called "bugs," can be difficult to find and fix. A debugging aid called ODT (On-line Debugging Technique) is described in Chapter 14. You will use it to correct the program's final error and to rerun the program. For now, however, the error will be pointed out and explained.

Look at line 40 in the assembly listing. Notice that the instruction in this line converts a digit into the appropriate ASCII code before printing it on the terminal. To do this, the constant 10 is added into the value of the digit already stored in memory, and then the value is converted — via '0, the ASCII code for 0 — to an ASCII code that can be printed. Unless you explicitly designate a value as decimal, however, the assembler assumes all values used in the program are octal. Therefore, it interprets the constant as 10(octal) or 8(decimal), and adds the wrong value every time. The conversion consequently causes the codes of the ASCII characters / and . to be used as results in some cases. The codes of other digits, while representing numeric values, are also off by two. To correct this error, you must insert a period after the 10 in line 40. The period instructs the assembler to accept the constant value 10 as a decimal value.

## **COMBINING OPERATIONS**

**EXECUTE**

To produce program results, you first assembled the MACRO-11 source program, SUM.MAC, then linked it, and finally ran the resulting .SAV file, SUM.SAV. You can combine these three operations using one monitor command, the EXECUTE command.

### **NOTE**

In order to use the EXECUTE command, the following files must be present on your system volume:

SUM.MAC  
MACRO.SAV  
LINK.SAV  
SYSLIB.OBJ

The last file, SYSLIB.OBJ, is required only if the MACRO-11 program you need to link refers to routines that are contained in the system library. The program used in this demonstration, SUM.MAC, does not require SYSLIB.OBJ.

The EXECUTE command instructs the system to select the appropriate language processor, then process, link, and run the program. There are several ways to establish which language processor the EXECUTE command invokes. One way is to specify a language-name option, such as /MACRO, which invokes the MACRO-11 assembler. Another way is to omit the language-name option and explicitly specify the file type for the source file. The EXECUTE command then invokes the language processor that corresponds to that file type. Specifying the file SUM.MAC, for example, invokes the MACRO-11 assembler. A third way to establish the language processor is to let the system choose a file type of .MAC, .DBL, or .FOR for the source file you name. If, for example, you specify the file SUM,

the monitor searches device SY: (your system device) for the files SUM.MAC, SUM.DBL, and SUM.FOR, in that order. If it finds a file named SUM.MAC, it invokes the MACRO-11 assembler to process the file. For example, to combine the assemble-link-run operations you performed in this chapter, you use the following command:

### Long Command Format

```
.EXECUTE␣  
Files? SUM/LIST/CROSSREFERENCE␣
```

### Short Command Format

```
.EXECUTE SUM/LIST/CROSSREFERENCE␣  
THE VALUE OF E IS:  
2.5/606/606237.2301314.06525/130440275535025.71477737352744745405502.544
```

Notice how you use the /LIST and /CROSSREFERENCE options following the file name to request both an assembly and a cross-referenced listing.

#### EXECUTE

Combine the assemble-link-run operations into one command.

#### EXECUTE file/MACRO

Combine the process-link-run operations into one command, and specify the input file to be a MACRO-11 file.

#### EXECUTE/CROSSREFERENCE

Produce a cross-referenced listing file.

#### EXECUTE/LIST

Produce a listing file of the source program.

#### LINK

Link individual object modules together to form a complete program and produce a load module.

#### MACRO

Assemble the MACRO-11 source program, and produce an object module.

#### MACRO/CROSSREFERENCE

Assemble the MACRO-11 source program, and produce both an object module and a cross-referenced listing file.

#### MACRO/LIST

Assemble the MACRO-11 source program, and produce both a listing on the line printer and an object module.

#### RUN

Run the indicated load module.

**SUMMARY:  
COMMANDS TO  
RUN MACRO-11  
PROGRAMS**

## FILE MAINTENANCE

Before continuing, you should perform the necessary file maintenance operations. Obtain a directory of all files on your system volume that have the name SUM, regardless of file type; these files were created as a result of the exercises in this chapter.

### Long and Short Command Format

```
.DIRECTORY SUM.*  
08-Jan-83  
SUM .BAK 3 08-Jan-83 SUM .SAV 2 08-Jan-83  
SUM .LST 9 08-Jan-83 SUM .MAC 3 08-Jan-83  
SUM .OBJ 1 08-Jan-83  
5 Files, 18 Blocks  
480 Free blocks
```

The fact that you have corrected errors in the source file of SUM.MAC makes the version of that file on your storage volume obsolete. Therefore, transfer the updated copy from your system volume back to VOL:, replacing the copy of SUM.MAC on the storage volume with the new version.

### Long Command Format

```
.COPY  
From? SUM.MAC  
To ? VOL:SUM.MAC
```

### Short Command Format

```
.COPY SUM.MAC VOL:SUM.MAC
```

Similarly, transfer SUM.SAV and SUM.OBJ to your storage volume. This allows you to rerun the MACRO-11 program without reassembling and relinking the source.

### Long Command Format

```
.COPY  
From? SUM.SAV,SUM.OBJ  
To ? VOL:  
Files copied:  
DK:SUM.SAV to VOL:SUM.SAV  
DK:SUM.OBJ to VOL:SUM.OBJ
```

### Short Command Format

```
.COPY SUM.SAV,SUM.OBJ VOL:  
Files copied:  
DK:SUM.SAV to VOL:SUM.SAV  
DK:SUM.OBJ to VOL:SUM.OBJ
```

Once you have transferred to your storage volume the files you want saved, delete from the system volume those you no longer need (that is, all the SUM files).

## Long Command Format

```
.DELETE/NOQUERYRET
Files? SUM.*RET
```

## Short Command Format

```
.DELETE/NOQUERY SUM.*RET
```

Notice that the /NOQUERY option suppresses confirmation when wildcard construction is used.

Finally, obtain an up-to-date directory listing of your storage volume so that you can see its current status.

## Long and Short Command Formats

```
.DIRECTORY VOL:RET
08-Jan-83
SUM .SAV 2 08-Jan-83 SUM .OBJ 1 08-Jan-83
EXAMP .FDR 2 26-Aug-82 EXAMP .MAC 3 26-Aug-82
GRAPH .FDR 2 08-Jan-83 GRAPH .LST 8 08-Jan-83
GRAPH .OBJ 16 08-Jan-83 GRAPH .SAV 21 08-Jan-83
SUM .MAC 3 08-Jan-83 MATCH .BAS 3 08-Jan-83
10 Files, 61 Blocks
4701 Free blocks
```

This completes the MACRO-11 demonstration. Continue now to Chapter 12 to learn more about the link operation.

*PDP-11 MACRO-11 Language Reference Manual* (AA-5075C-TC). Maynard, Mass.: Digital Equipment Corporation, 1983.

A reference manual for the PDP-11 programmer using the MACRO-11 assembly language.

*PDP-11 Peripherals Handbook*. Maynard, Mass.: Digital Equipment Corporation, 1981-82.

A technical description of the PDP-11 peripheral devices, including necessary programming information.

*PDP-11 Processor Handbook*. Maynard, Mass.: Digital Equipment Corporation, 1981.

A technical description of the various PDP-11 processors, including complete information concerning the PDP-11 instruction set.

*PDP-11 Programming Card*. Maynard, Mass.: Digital Equipment Corporation, 1975.

A pocket-sized folding card summary of PDP-11 machine instructions used by the various PDP-11 assembly language processors.

*PDP-11 Software Handbook* (EB-21759-20). Maynard, Mass.: Digital Equipment Corporation, 1982-83.

A general overview and introduction to available PDP-11 software, operation systems, and language processors.

## REFERENCES

*Running a MACRO-11 Assembly Language Program*

*RT-11 Programmer's Reference Manual* (AA-H378B-TC). Maynard, Mass.: Digital Equipment Corporation, 1983.

An RT-11 system-specific programming manual for the MACRO-11 programmer.

*RT-11 System User's Guide* (AA-5279C-TC). Maynard, Mass.: Digital Equipment Corporation, 1983.

A guide to the use of the RT-11 operating system.

*RT-11 System Utilities Manual* (AA-M239A-TC). Maynard, Mass.: Digital Equipment Corporation, 1983.

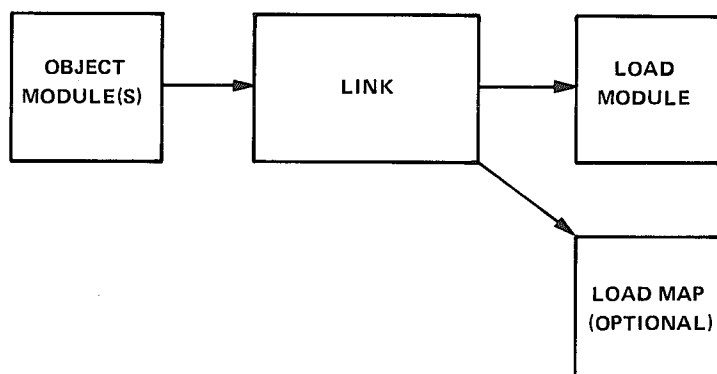
A guide to the use of the RT-11 System Utilities.

## CHAPTER 12

### LINKING OBJECT PROGRAMS

Programs that you write in the MACRO-11 and FORTRAN IV programming languages require additional processing after their conversion to object format. Before you can run these programs on the system, you must link them. The link operation:

- Joins together the object modules that use a symbol with the object module that defines it.
- Relocates individual object modules as necessary and assigns absolute (permanent) memory addresses; it can also define an overlay structure.
- Produces a load module and an optional load map (Figure 12-1).



**Figure 12-1 Link Functions**

Program linking gives you the advantage of a modular approach to programming. You can create an entire program as a series of smaller, independent subprograms. One of these is written as the main, or controlling, program, and the rest as subordinate subprograms and subroutines. You use a language processor to translate each part of the program into an object module. Then you use the linker to join all the object modules together into a complete, functioning unit.

Modular programming makes program creation and debugging easier. For example, several programmers can simultaneously work on a single program, each creating a portion of it. The individual portions, or subprograms, can be processed and linked with test programs and debugged for logic errors separately. Then all the object modules can be joined together to form a complete program that can be tested as a whole. If errors occur at this stage, only those object modules with errors need be debugged and changed.

In addition, modular programming allows you to make use of library files. These are files containing subprograms and subroutines that have been debugged. After you join library files with your program at link-time, their routines can be used by your program as needed.

## RESOLVING SYMBOLIC AND LIBRARY REFERENCES

The linker reads through all the object modules that you indicate as input to the LINK command. It gathers and evaluates information (provided to the modules by the language processor) that is necessary for program linking. For each input module, this information includes the object code, information needed for relocation, the relative address of the first instruction, the global symbols used, and the absolute length of each program and program section.

One of the linker's first functions is to resolve all user-defined symbolic references and library references in the joined routines. There are two types of user-defined symbols — internal symbols and global symbols.

Internal symbols are limited to the object module in which they appear; thus, they cannot be referenced from or defined in any other module. A program containing only internal symbolic references — like those in the demonstration program in Chapter 11 — is complete in itself and does not need to be joined with any other object programs at link-time. Thus, internal symbols are not resolved at link-time because they have already been resolved by the language processor.

Global symbols, on the other hand, are the key to modular programming. Global symbols provide the communication between object modules. Such symbols may be symbolic labels to instructions, symbolic labels to data, or symbols that are equated to a value or constant. Global symbols are defined in one object module and referenced from other object modules that have been separately assembled or compiled. Such symbols must be designated as global in the source code. The following segment of MACRO-11 assembly language code illustrates the use of global symbols:

```
.MAIN. MACRO V05.00 SATURDAY 08-JAN-83 08:42 PAGE 1

1          .GLOBL A,C,VALUE          !DECLARE A, C, AND VALUE
2          !AS GLOBAL SYMBOLS
3 000000 013500          A: MOV B(R5)+,R0 !GLOBAL SYMBOL A IS DEFINED
4          !HERE AND CAN BE REFERENCED
5          !FROM OTHER MODULES, PROBABLY
6          !BY A SUBROUTINE CALL
7 000002 016701 000018          MOV LOCAL,R1 !LOCAL IS AN INTERNAL SYMBOL
8          !DEFINED AND REFERENCED ONLY
9          !WITHIN THIS MODULE
AU 10 000006 000000 000007 000000G  USR PC,C !CALL TO GLOBAL ROUTINE C,
11          !DEFINED IN ANOTHER MODULE
12 000014 013501          MOV B(R5)+,R1
13 000016 005003          CLR R3
14 000020 000207          RTS PC
15 000022 000011          VALUE: .WORD 11 !GLOBAL SYMBOL VALUE IS USED TO
16          !REFERERENCE THIS DATA LOCATION
17 000024 177777          LOCAL: .WORD 177777 !INTERNAL SYMBOL USED FOR DATA
18          .END
```

While internal symbolic references, such as LOCAL in the example, can be resolved by the assembler or compiler within the single program unit, global references, such as C, cannot. They require other object modules. During translation, the language processor notes in the object module those symbols that are global. During linking, the linker keeps track of the global references and definitions found in all the object modules. As linking proceeds, it makes the appropriate correlations and modifies instructions or data as necessary. After linking, the linker prints on the terminal a list of all symbolic references that were not resolved (undefined globals), either because of a programming error or because all necessary object modules were not included in the linking process.

References to library files also involve the use of global symbols. You access the routines in a library by naming a routine as a global symbol in the source code of your program. You then link your program with the appropriate library file, and the linker resolves the library references just as it does any global symbol. Library usage is discussed in greater detail in Chapter 13.

A second important function of the linker is to “fix” the relative memory addresses so that they are absolute.<sup>1</sup> The object module represents translated source instructions that have been assigned memory addresses relative to a base address of 0.

Look back at the assembly listing in Chapter 11. Note the second column; these addresses are relative to a base address of 0. Thus the first instruction is assembled at relative address 0, the second at relative address 6, and so on. A program cannot actually be stored and run in memory using locations relative to address 0, however, because system information is already stored in some of these locations. For example, the RT-11 operating system uses byte addresses 40 through 57 to store information about the program currently executing. In addition, the RT-11 operating system uses locations in the upper range of memory for storing the resident monitor. Thus, the linker must assign memory addresses to your program that are not already in use or that will not be used during program execution. It must, therefore, assign absolute memory addresses to the relative addresses assigned by the language processor.

The linker normally starts assigning memory addresses at address 1000, since this begins a large section of free memory

<sup>1</sup>FORTTRAN IV and BASIC-11 users who have not performed the demonstration in Chapter 11 may wish to read the section in that chapter entitled The MACRO-11 Language Processor. That section explains the concept of converting and storing instructions in computer memory.

## **PROGRAM RELOCATION AND ADDRESS ASSIGNMENT**

space. So, to obtain the actual addresses used for program loading, you must add the relocation constant 1000 to each relative address shown in the assembly listing.

A conflict arises when several individually processed object modules are linked together. The linker cannot assign memory addresses starting at 1000 to every module, since address assignments of one would then override those of another. However, part of the information that the language processor calculates and passes to the linker is the size of each program section in each module. So the linker simply adds this size into the relocation constant for each module and assigns higher addresses, appropriately modifying the relative location of all instructions and data as necessary to account for the relocation of each individual module. Figure 12-2 illustrates the relocation that must occur to accommodate object modules linked together.<sup>1</sup>

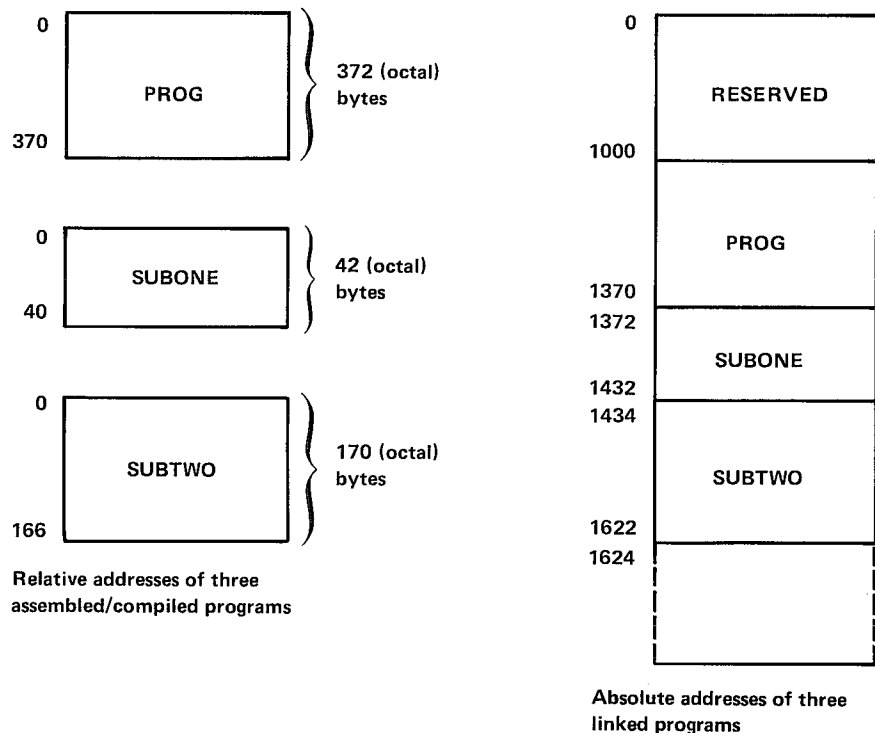


Figure 12-2 Object Module Relocation

**ABSOLUTE AND RELOCATABLE PROGRAM SECTIONS**

Just as global symbols allow you to create an entire program, using several individual object modules, program sections allow you to create an object module as a series of individual sections. The advantages gained through the sectioning of programs relate primarily to control of memory allocation, program modu-

<sup>1</sup>A load map for this relocation example is shown later in the chapter.

larity, and more effective partitioning of memory. The linker processes the program section information in the object modules as directions on how to create the executable program image.

The FORTRAN IV and MACRO-11 language processors insert program sectioning information into the object module. The FORTRAN IV language processor does this automatically when program sectioning is implied by the source language statements in a user program. For example, FUNCTION, SUBROUTINE, and COMMON statements result in the production of program section directives. In MACRO-11 assembly language, you are responsible for explicitly directing the assembler to output program section information for the linker. You do this through the .PSECT (or .CSECT and .ASECT) MACRO-11 assembly language statement.

Some of the basic functions associated with program sections are:

1. Instructions or data can be placed in absolute locations in memory. The named absolute program section (. ABS.) allows you to instruct the linker as to exactly where to place program code or data. Declaring a section as part of the absolute program section instructs the assembler or compiler to use the internal value of the program counter as the physical memory address to be assigned after linking. This section is processed relative to absolute memory address 0 and is not relocated at link time.
2. Named relocatable program sections are used to group data or instructions into logical portions of memory. The FORTRAN IV COMMON statement invokes this construct to allow access to named data areas from many separate routines. Declaring a section as part of a named relocatable program section causes the section to be processed at relocatable address 0. Such sections are relocated by the linker.
3. If you do not care about having exact control over where a portion (section) of a program will be placed in memory, use the blank program section — a special program section that the linker treats as relocatable. The linker decides where to place this program section in the loadable memory image. The blank program section is the default for a MACRO-11 source program and remains in effect until an explicit program section is identified (the program example in Chapter 11 used the blank program section).
4. A program section can be identified as an instruction section. The linker, using this information, can provide automatic loading of declared overlay code when needed by the executing program (this will be discussed in more detail).

The language processor, then, actually maintains several program counters — one for the absolute program section, one for the unnamed relocatable program section, and as many as needed (maximum is 254) for named relocatable program sections. The assembled example that follows helps explain this concept.

```
.MAIN, MACRO V05.00 SATURDAY 08-JAN-83 09:04 PAGE 1

1
2
3
4 000000 005000          START: CLR R0
5 000002 012701 000034'   MOV #BEG,R1
6 000006 082100          LOOP: ADD (R1)+,R0
7 000010 022701 000044'   CMP #BEG+10,R1
8 000014 100374          BPL LOOP
9 000018 012787 002000 000020 MOV #2000,ADDR
10 000024 005003          CLR R3
11 000000          .PSECT CLEAR          INAMED RELOCATABLE PROGRAM
12 000000 012703 000100   MOV #100,R3          ISECTION IS DECLARED (VIA ".PSECT NAME")
13 000004 012701 000044'   MOV #ADDR,R1
14 000010 005021          AGAIN: CLR (R1)+
15 000012 005303          DEC R3
16 000014 001375          BNE AGAIN
17 000000          .ASECT
18 000002          .,42          IABSOLUTE PROGRAM SECTION
19 000042 001000          .WORD 1000          IDECLARED (VIA ".ASECT")
20
21
22 000026          .PSECT          ITHE VALUE 1000 WILL BE
23 000026 005267 000012   INC ADDR          ISTORED IN ABSOLUTE MEMORY LOCATIN 42
24 000032 000000          HALT          IWHEN THE PROGRAM IS EXECUTED
25 000034 000001 000002 000003 BEG: .WORD 1,2,3,4          IBACK TO UNNAMED RELOCATABLE
26 000042 000004          .WORD 0          IPROGRAM SECTION
27 000044 000000          INOTE THAT YOU CAN WRITE LANGUAGE STATEMENTS THAT WILL BE LOADED
28          ICONTIGUOUSLY IN MEMORY, BUT DO NOT NECESSARILY OCCUR CONTIGUOUSLY
29          IIN THE SOURCE PROGRAM (I.E., THE CODE AT LINES 1-10 AND 22-29)
30          .END
```

Since the system does not know at assembly (or compile) time into which actual memory locations each relocatable section goes, all references among sections (see line 18) are relative to the base of the section. This information is then passed to the linker so that it can make the appropriate adjustments at link-time.

## The Overlay Feature

The RT-11 linker is also capable of handling the special relocation and address assignments that are required whenever you indicate that an overlay structure is needed. An overlay structure is necessary when you write a program that is too large to fit in the available memory of your system. You write the program in discrete parts (some programming restrictions must be observed) so that your program can subsequently be executed in parts. Some of these parts, or segments, are allowed to share memory with other segments, thus reducing the overall memory requirements of the program. One segment of the program is called the root segment and must remain in memory at all times. The root segment contains the necessary information for use by the other segments of the program, called overlay segments. Overlay segments are stored on storage volumes and brought into memory as needed. The purpose of the overlay structure is for parts of the program to share the available memory in such a way that when one part is complete, it is overlaid (and therefore erased) by another.

You indicate how to plan to overlay your program by using the /PROMPT option in the LINK command line. The linker then

creates a load module that contains the necessary information for loading the appropriate segments as needed during execution. The *RT-11 System Utilities Manual* explains the overlay feature in more detail. You need not specify an overlay structure for the examples demonstrated in this chapter.

The load module is the result of the linking processes described so far: joining object modules, resolving symbolic and library references, relocating object modules, assigning absolute addresses, and creating an overlay structure if required. The load map is essentially a synopsis of the load module — that is, what memory looks like when the program is loaded and ready to be executed.

In Chapters 9 and 11, you produced load modules, but you did not request load maps. You obtain a load map by using the /MAP option with the LINK (or EXECUTE) command. At this time, relink the FORTRAN IV or MACRO-11 object module that you stored on VOL: and use the /MAP option to produce a load map.<sup>1</sup> The load map is created as a file on the system volume, which is the default storage volume for input/output operations. The load map has the name of the first input module and a file type of .MAP.

## PRODUCING A LOAD MODULE AND A LOAD MAP

/MAP

### Long Command Format

MACRO-11 object module:

```
.LINK (RET)  
Files? VOL:SUM/MAP(RET)
```

FORTRAN IV object module, if FORLIB is not included in SYSLIB:

```
.LINK (RET)  
Files? SYSLIB, FORLIB, VOL:GRAPH/MAP(RET)
```

FORTRAN IV object module, if FORLIB is included in SYSLIB:

```
.LINK (RET)  
Files? VOL:GRAPH/MAP(RET)
```

### Short Command Format

MACRO-11 object module:

```
.LINK VOL:SUM/MAP(RET)
```

<sup>1</sup>FORTRAN IV users who followed the special instructions in Appendix B for loading the LINK volume should check that this volume is loaded in device unit 0. FORTRAN IV users who have a special FORTRAN IV language volume, but not a LINK volume, should make sure that the FORTRAN IV volume is now loaded in device unit 0.

FORTRAN IV object module, if FORLIB is not included in SYSLIB:

```
.LINK SYSLIB, FORLIB, VOL:GRAPH/MAPRET
```

FORTRAN IV object module, if FORLIB is included in SYSLIB:

```
.LINK VOL:GRAPH/MAPRET
```

Now list the .MAP file on either the line printer or terminal, choosing the appropriate command:

### Long Command Format

(Line printer)

(Terminal)

MACRO-11 object module:

```
.PRINTRET
Files? SUM.MAPRET
```

```
.TYPERET
Files? SUM.MAPRET
```

FORTRAN IV object module:

```
.PRINTRET
Files? GRAPH.MAPRET
```

```
.TYPERET
Files? GRAPH.MAPRET
```

### Short Command Format

(Line printer)

(Terminal)

MACRO-11 object module:

```
.PRINT SUM.MAPRET
```

```
.TYPE SUM.MAPRET
```

FORTRAN IV object module:

```
.PRINT GRAPH.MAPRET
```

```
.TYPE GRAPH.MAPRET
```

For your convenience, both maps are provided here. In addition, a load map of the relocation example used in Figure 12-2 is also provided.

```
RT-11 LINK V08.00      Load Map      Saturday 08-Jan-83 10:00 Page 1
SUM .SAV      Title:  SUM.MA Ident:
Section Addr  Size  Global Value  Global Value  Global Value
. ABS.  000000 001000 = 256. words (RW,I,GBL,ABS,OVR)
        001000 000372 = 125. words (RW,I,LCL,REL,CON)

Transfer address = 001000, High limit = 001370 = 380. words
RT-11 LINK V08.00      Load Map      Saturday 08-Jan-83 10:11 Page 1
SYSLIB.SAV Title:  .MAIN. Ident: FORV02
Section Addr  Size  Global Value  Global Value  Global Value
. ABS.  000000 001000 = 256. words (RW,I,GBL,ABS,OVR)
        $USRSW 000000 $RF2A1 000000 $HRDWR 000000
        .VIR  000000 $NLCHN 000006 $SYSV$ 000012
        $WASIZ 000152 $LRECL 000210 $TRACE 004737
```

```

OTS$I 001000 017722 = 4073. words (RW,I,LCL,REL,CON)
$*OTSI 001000 $CVTIF 001000 $CVTIC 001014
$CVTID 001014 CCI$ 001026 CDI$ 001026
$IC 001026 $ID 001026 CFI$ 001042
$IR 001042 EXP 001126 ADF$IS 001466
ADF$PS 001474 SUF$PS 001500 SUF$MS 001504
ADF$MS 001516 SUF$IS 001526 $ADDF 001534
$SUBF 001550 SUF$SS 001562 $SBR 001562
ADF$SS 001566 $ADR 001566 ADD$ 001602
DIF$PS 002226 DIF$MS 002232 DIF$IS 002242
$DIVF 002250 DIF$SS 002262 $DVR 002262
MUF$PS 002550 MUF$MS 002554 MUF$IS 002564
$MULF 002572 MUF$SS 002604 $MLR 002604
$OTI 003142 $*OTI 003144 $SETOP 003354
$$SET 005026 SQRT 005322 STK$L 005516
STK$I 005522 STK$F 005526 IOR$ 005536
AND$ 005542 EQV$ 005550 XOR$ 005552
NMI$IM 005566 NMI$II 005600 BLE$ 005610
BEQ$ 005612 BGT$ 005620 BGE$ 005622
BRA$ 005624 BNE$ 005630 BLT$ 005632
CAI$ 005642 CAL$ 005650 END$ 005700
ERR$ 005712 $END 005724 $ERR 005742
$OPNER 005764 $CHKER 006022 $IOEXI 006046
$EOL 006114 EOL$ 006116 EXIT 006232
MOF$SS 006236 MOF$MS 006250 MOF$PS 006262
MOF$SM 006266 MOF$SP 006276 MOF$MM 006302
MOF$MA 006314 MOF$MP 006322 MOF$PM 006330
MOF$PA 006334 MOF$PP 006340 MOF$RS 006344
MOF$RM 006352 MOF$RA 006362 MOF$RP 006366
NGD$S 006372 NGF$S 006372 NGD$M 006404
NGF$M 006404 NGD$P 006420 NGF$P 006420
NGD$A 006424 NGF$A 006424 ADI$SS 006430
ADI$SA 006434 ADI$SM 006440 ADI$IS 006444
ADI$IA 006450 ADI$IM 006454 ADI$MS 006460
ADI$MA 006464 ADI$MM 006470 CMI$SS 006474
CMI$SI 006500 CMI$SM 006504 CMI$IS 006510
CMI$II 006514 CMI$IM 006520 CMI$MS 006524
CMI$MI 006530 CMI$MM 006534 IFW$ 006540
$IFW 006544 $$IFW 006550 IFW$$ 006606
MOI$RS 006656 MOL$RS 006656 MOI$RM 006662
MOI$RF 006666 MOI$RA 006670 MOI$SS 006674
MOL$SS 006674 MOI$SM 006700 MOI$SA 006704
MOI$IS 006710 MOL$IS 006710 REL$ 006710
MOI$IM 006714 MOI$IA 006720 MOI$MS 006724
MOI$MM 006730 MOI$MA 006734 MOI$OS 006740
MOI$OM 006744 MOI$OA 006750 MOI$IS 006754
MOI$IM 006762 MOI$IA 006770 ICI$S 006776
ICI$M 007002 ICI$P 007006 ICI$A 007010
DCI$S 007014 DCI$M 007020 DCI$P 007024
DCI$A 007026 IDINT 007032 INT 007032
MOI$IP 007060 MOI$SP 007062 MOI$PP 007070

```

```

RT-11 LINK V08.00 Load Map Saturday 08-Jan-83 10:11 Page 2
SYSLIB.SAV Title: .MAIN. Ident: FORV02

```

```

MOI$MP 007074 MOI$PS 007104 MOI$FM 007112
MOI$PA 007120 MOI$OP 007126 MOI$IP 007134
ISN$ 007144 $ISNTR 007150 LSN$ 007164
$LSNTR 007170 SUI$SS 007324 SUI$SA 007330
SUI$SM 007334 SUI$IS 007340 SUI$IA 007344
SUI$IM 007350 SUI$MS 007354 SUI$MA 007360
SUI$MM 007364 MOL$SM 007370 MOL$SA 007374
MOL$MS 007400 MOL$MM 007410 MOL$MA 007414
MOL$SP 007420 MOL$FP 007426 MOL$MP 007432
MOL$FM 007442 MOL$PS 007450 MOL$PA 007454
MOL$IM 007462 MOL$IA 007470 MOL$IP 007476
LLE$ 007506 LEG$ 007510 LGT$ 007516
LGE$ 007520 LNE$ 007530 LLT$ 007532
TSL$S 007536 TSL$M 007542 TSL$I 007546
TSL$P 007554 MAXO 007562 MINO 007606
RET$L 007632 RET$F 007636 RET$I 007644
RET$ 007646 $OTIS 007702 $*OTIS 007704
TVL$ 010024 $TVL 010024 TVF$ 010032
$TVF 010032 TVD$ 010040 $TVD 010040
TVR$ 010046 $TVR 010046 TVP$ 010054
$TVP 010054 TVI$ 010062 $TVI 010062
SAL$IM 010216 SAL$SM 010220 SVL$IM 010224
SVL$SM 010226 SAL$MM 010234 SVL$MM 010240
SAL$IP 010244 SAL$SP 010246 SVL$IP 010252
SVL$SP 010254 SAL$MP 010262 SVL$MP 010266
$CVTFB 010272 $CVTFI 010272 $CVTCB 010306
$CVTCI 010306 $CVTDB 010306 $CVTDI 010306
CIC$ 010320 CID$ 010320 CLC$ 010320
CLD$ 010320 $DI 010320 CIF$ 010330
CLF$ 010330 $RI 010330 CIL$ 010442
CLI$ 010446 $INITI 010450 $CLOSE 010566
$ERRTB 011344 $ERRS 011451 $FCHNL 015212

```

```

                                $FIO 016054 $$FIO 016060 $PUTRE 017224
                                $PUTBL 017532 $GETBL 017742 $EOFIL 020126
                                $EOF2 020142 SAVRG$ 020162 THRD$ 020340
                                $STFS 020342 STP$ 020350 $STP 020350
                                FOO$ 020354 $EXIT 020374 $WAIT 020520
                                $VRINT 020562 $DUMPL 020574
DTS$F 020722 000054 = 22. words (RW,D,GBL,REL,DVR)
SYS$I 020776 000244 = 82. words (RW,I,LCL,REL,CON)
                                LEN 020776 REPEAT 021014 SCOPY 021146
USER$I 021242 000000 = 0. words (RW,I,LCL,REL,CON)
$CODE 021242 001316 = 359. words (RW,I,LCL,REL,CON)
DTS$D 022560 001036 = 271. words (RW,I,LCL,REL,CON)
                                $$DTSC 021242 FUN 022120 PUTSTR 022266
                                $$DTSO 022560 $OPEN 022560
SYS$D 023616 000000 = 0. words (RW,I,LCL,REL,CON)
$DATAF 023616 000106 = 35. words (RW,D,LCL,REL,CON)
DTS$D 023724 000006 = 3. words (RW,D,LCL,REL,CON)
                                NHCLN$ 023730
DTS$S 023732 000002 = 1. words (RW,D,LCL,REL,CON)
                                $ADTS 023732
SYS$S 023734 000004 = 2. words (RW,D,LCL,REL,CON)
                                $SYSLB 023734 $LOCK 023736 $CRASH 023737
$DATA 023740 000536 = 175. words (RW,D,LCL,REL,CON)
USER$D 024476 000000 = 0. words (RW,D,LCL,REL,CON)
.$$$$. 024476 000000 = 0. words (RW,D,GBL,REL,DVR)

RT-11 LINK VOB.00 Load Map Saturday 08-Jan-83 10:11 Page 3
SYSLIB.SAV Title: .MAIN. Ident: FORV02

```

```

Transfer address = 021242, High limit = 024474 = 5278. words

RT-11 LINK VOB.00 Load Map Saturday 08-Jan-83 10:16 Page 1
TEST .SAV Title: .MAIN. Ident:

Section Addr Size Global Value Global Value Global Value
. ABS. 000000 001000 = 256. words (RW,I,GBL,ABS,DVR)
      001000 000626 = 203. words (RW,I,LCL,REL,CON)
      PROG 001000 SUBONE 001372 SUBTWO 001434

Transfer address = 001000, High limit = 001624 = 458. words

```

The second line has the name and file type of the load module created. Next, the absolute section and each named and unnamed section are listed under the SECTION column. To the right are abbreviated codes designating whether the section contains Instructions or Data, is Read/Write or Read Only, is a Local or Global section, is Relocatable or Absolute, is Concatenated or Overlaid. Below this falls a listing of all the global symbols (GLOBAL) and their values (VALUE). Finally, at the end of the map is the transfer address where the program actually starts when executed, followed by the high limit — the total number of bytes used by all the individual program sections.

Look first at the MACRO-11 load map. The default absolute section starts at absolute location 0; its size is 1000 bytes. Thus, it extends from absolute memory location 0 through absolute memory location 777. The unnamed program section (there were no named program sections in this program) starts at absolute 1000; its size is 372 bytes. Thus it extends from absolute location 1000 to absolute location 1370. The high limit of this program (total bytes) is therefore 1370. Since this program is not linked to any other object modules, there are no global symbols and the rest of the map is blank.

Look now at the FORTRAN IV load map, remembering that it reflects the appropriate expansions into machine language code provided by the FORTRAN IV compiler. Again, the absolute

section extends from absolute 0 through absolute 777. Globals listed in the absolute section show the global variable names that the program uses as constants throughout the program.

The unnamed relocatable program section begins at absolute location 1000. Some of the named relocatable sections that are declared are OTS\$P, SYS\$I, and \$CODE. Global symbols and their respective addresses appear to the right of all sections. The total number of bytes used is 24474, or 5278(decimal) words.

The third load map is for the program illustrated in Figure 12-2. First, the map shows the absolute program section, labeled .ABS. It extends from location 0 through location 777. Next, the map shows the unnamed program section, which begins at location 1000 and is 1624 bytes long. This program section consists of a main program, called PROG, and the subroutines SUBONE and SUBTWO that were linked with PROG. Look again at Figure 12-2 to see how these routines fit into memory. The transfer, or starting, address is 1000, and the total number of bytes the program occupies is 1624, or 458(decimal) words.

Load maps are most helpful when used in debugging to locate and correct assembly language programming errors. Load maps are not generally obtained or used for FORTRAN IV programs, except to determine program size. In Chapter 14 you will see how a load map is used to debug the one remaining error in the MACRO-11 demonstration program.

#### LINK

Link individual object modules together to form a complete program and to produce a load module.

#### LINK/MAP

Link individual object modules, and produce a load map showing all address assignments made during linking.

#### NOTE

FORTTRAN IV users who followed the special instructions in Appendix B to load the language or LINK volume should now stop the system, unload that volume, load the main system volume, and rebootstrap the system before going on to Chapter 13.

*RT-11 System Utilities Manual (AA-M239A-TC)*. Maynard, Mass.: Digital Equipment Corporation, 1983.

A guide to the use of the RT-11 system utilities.

**SUMMARY:  
COMMANDS  
FOR LINKING  
PROGRAMS**

**REFERENCE**



## CHAPTER 13 CONSTRUCTING LIBRARY FILES

A library is a specially constructed file consisting of one or more programming routines. Generally, these routines provide services that you are apt to need repeatedly, or services that are related and so have been gathered together for ease in use and storage. You use the routines in a library by joining the library file with your source program. Usually this occurs at link-time; but in the case of assembly language programs, it may also occur at assembly-time.

The RT-11 operating system provides several library files; SYSLIB and VTLIB for example. These libraries supply the monitor services, input and output routines, conversion routines, and other programming services that user programs may need. You can create other library files yourself. Thus you can construct libraries that contain routines specific to your programming needs or to the combined needs of those using your RT-11 system.

There are two kinds of library files — macro libraries and object libraries.

### KINDS OF LIBRARY FILES

Macro libraries, such as SYSMAC.SML, are used by MACRO-11 source programs at assembly-time and consist entirely of macros. A macro is described in Chapter 11 as a recurring sequence of coding instructions, which, when defined in a .MACRO statement, can then be called and used anywhere in your program. A macro library is merely several macro definitions gathered together into a single file. To use the macros in a macro library, you simply name those macros you plan to use in a .MCALL statement. When the assembler encounters the .MCALL statement during processing, it searches the appropriate macro library (SYSMAC.SML is default) for the definitions. It takes the definitions from the library and inserts them in a special table called the macro symbol table where they become available for use during assembly.

### Macro Libraries

Object libraries, such as SYSLIB, are used by assembled MACRO-11 source programs and/or by compiled FORTRAN IV source programs at link-time. These libraries consist of object modules that contain global routines; such routines have been defined with global entry points and then named as global symbols in the source program. During the link operation, the linker searches the object libraries to determine if they provide

### Object Libraries

definitions for any undefined globals. If the linker finds definitions, it takes those object modules containing the definition from the library and includes them in the link.

A special table, called the global symbol table, lists each global in a given object library. You can print this list on the terminal or the line printer and thus keep track of an object library's current contents.

## **CREATING AND MAINTAINING A LIBRARY FILE**

You create a library file by combining several macro routines, or several object modules, into a single larger file using the monitor LIBRARY command.

To build a macro library, first use the editor to create an ASCII text file that contains all the macro definitions. Then process this file using the LIBRARY command in combination with its /MACRO option. To update a macro library (that is, to add or delete macro definitions), simply edit the ASCII text file and then reprocess the file with the LIBRARY command.

To build an object library, use the editor to create an ASCII text file. The file contains the routines and functions written as complete program segments in either the MACRO-11 assembly language or the FORTRAN IV programming language. Then process the file, producing an object module. Next, use the LIBRARY command in combination with its /CREATE option. Once the library file is created, update it (add and delete routines) by means of various other options to the LIBRARY command.

In the following exercises, you create an object library that contains three input object modules. The routines in two of these modules can be used by both MACRO-11 and FORTRAN IV programs; the routine in the third module can be used only by FORTRAN IV programs.

To build the library file, first use the editor to create the three ASCII text files. Then convert the ASCII text files to object format. Finally, process the object files with the LIBRARY command. Once you create the library files, use LIBRARY command operations and options to add and delete modules and globals and to obtain a listing of the library file contents.

### **Creating Object Library Input Files**

The first step in building an object library is to prepare the source code of the routines and functions that you choose to include in the library. Use the editor to create the following three text files, calling them FIRST.MAC, SECOND.MAC, and THIRD.FOR, respectively. FORTRAN IV users should create all three files; MACRO-11 users (who do not use FORTRAN IV) should create only the first two files.

**FIRST.MAC**

```

        .TITLE    COMB
        .MCALL    .PRINT
;       I=LEN(A)
        .GLOBL   LEN
LEN:    TST      (R5)+    ;SKIP # OF ARGS
        MOV      @R5,R0  ;GET STRING POINTER
1$:    TSTB     (R0)+    ;FIND END OF STRING
        BNE     1$      ;LOOP UNTIL NULL BYTE
        DEC     R0      ;BACK UP
        SUB     @R5,R0  ;CALC # OF CHARS IN STRING
        RTS     PC
;
        CALL    PRINT(ISTRNG)
        .GLOBL  PRINT
PRINT:  MOV     2(R5),R0    ;ADDR OF ISTRNG
        .PRINT  ;.PRINT
        RTS     PC        ;AND RETURN
        .END

```

**SECOND.MAC**

```

        .TITLE    ITTOUR
;       I=ITTOUR(ICHAR)
;       I=0      CHARACTER HAS BEEN OUTPUT
;       =1      RING BUFFER IS FULL
        .MCALL    .TTOUTR
        .GLOBL   ITTOUR
ITTOUR:MOVB    @2(R5),R0    ;GET CHARACTER
        .TTOUTR  ;.TTOUTR
        BIC     R0,R0      ;CLEAR ERROR FLAG
        ADC     R0
        RTS     PC        ;RETURN
        .END

```

**THIRD.FOR**

```

C       CALL PUTSTR(LUN,AREA,CC)
        SUBROUTINE PUTSTR(LUN,AREA,CC)
        LOGICAL*1 AREA(250),CC
        IF(CC) GOTO 1
        WRITE (LUN,99)(AREA(I),I=1,LEN(AREA))
        RETURN
1       WRITE (LUN,99)CC,(AREA(I),I=1,LEN(AREA))
99      FORMAT(250A1)
        END

```

The routines in these files are representative of the kinds of services generally provided in a library file. They are, in fact, taken from the RT-11 system subroutine library, SYSLIB.

FIRST.MAC contains two global routines, LEN and PRINT. The LEN routine returns the number of characters in a string. PRINT outputs an ASCII string terminated with a zero byte to the terminal (it is the FORTRAN IV equivalent of the system macro .PRINT, used in the demonstration program in Chapter 11). SECOND.MAC contains one global routine, ITTOUR, which transfers a character to the console terminal. THIRD.FOR also contains one global routine, PUTSTR. This routine can be used only by FORTRAN IV programs and writes a variable-length character string on a specified FORTRAN IV logical unit (see GRAPH example).

Once you create these text files, the next step is to convert them from ASCII format to object format. Assemble or compile the text files as appropriate, first assembling FIRST.MAC and obtaining an object module (a listing is not necessary). FORTRAN IV users who are not familiar with the assembly process simply type the MACRO commands as shown.

### Long Command Format

```
.MACRO␣  
Files? FIRST␣
```

### Short Command Format

```
.MACRO FIRST␣
```

The command creates an object module called FIRST.OBJ on the system volume. If errors occur, the assembler prints a message on the terminal, indicating the number of errors encountered during assembly. No errors should occur.

In the same way, assemble SECOND.MAC. Again, no errors should occur.

### Long Command Format

```
.MACRO␣  
Files? SECOND␣
```

### Short Command Format

```
.MACRO SECOND␣
```

If any errors occur during the assembly operations, you have typed the source files incorrectly. Find and correct the typing errors, and reassemble.

If you are a FORTRAN IV user, continue by compiling THIRD.FOR.

## NOTE

If in Chapter 9 you needed to load the special FORTRAN/BASIC language volume, you must again load that volume before you can compile THIRD.FOR. Read Appendix B, Substituting Volumes During Operations, before continuing.

**Long Command Format**

```
.FORTRAN@  
Files? THIRD@  
PUTSTR
```

**Short Command Format**

```
.FORTRAN THIRD@  
PUTSTR
```

Notice that the compiler prints the name of the global (PUTSTR) generated. If any errors occur during the compile operation, you have typed the source file incorrectly. Find and correct the typing errors, and recompile.

Once you have produced the object modules, you are ready to build the object library file.

Use the LIBRARY command in combination with its /CREATE option to construct a library file. You must indicate in the command the name of the library file and the names of the input object modules. Call the library file LIBRA and specify as input the two object modules, FIRST and SECOND. The LIBRARY command assumes that the input modules have the .OBJ file type (unless you indicate otherwise) and automatically assigns .OBJ to the library file.

**Long Command Format**

```
.LIBRARY/CREATE@  
Library? LIBRA@  
Files ? FIRST,SECOND@
```

**Short Command Format**

```
.LIBRARY/CREATE LIBRA FIRST,SECOND@
```

Once the CREATE operation is complete, obtain a listing of the library file's contents, using the LIBRARY command with its LIST operation. The line printer is the assumed output device for the list file, although you may indicate a different output device by adding the two-letter device mnemonic to the LIST option that follows.

**Building the  
Object Library**

LIBRARY/ CREATE
--------------------

LIBRARY/LIST

**Long Command Format**

(Line printer)

(Terminal)

```
·LIBRARY/LISTRET
Library? LIBRARET
```

```
·LIBRARY/LIST:TT:RET
Library? LIBRARET
```

**Short Command Format**

(Line printer)

(Terminal)

```
·LIBRARY/LIST LIBRARET
```

```
·LIBRARY/LIST:TT: LIBRARET
```

The listing produced shows the library file's current contents. This library has three entry points: LEN and PRINT in the first module, and ITTOUR in the second module.

```
RT-11 LIBRARIAN V05.00 SAT 8-JAN-83 11:03:29
DK:LIBRA.OBJ SAT 8-JAN-83 10:59:43

MODULE      GLOBALS      GLOBALS      GLOBALS
              LEN          PRINT
              ITTOUR
```

**Updating the Object Library**

LIBRARY/  
INSERT

Once you have created an object library, you use various LIBRARY command operations to update and maintain it by adding and deleting modules and globals.

If you created the THIRD.OBJ object module, you can add it to the library file using the INSERT option. If you did not create this module, read through this section anyway; the command steps apply to any object module you wish to insert.

**Long Command Format**

```
·LIBRARY/INSERTRET
Library? LIBRARET
Files ? THIRDRET
```

**Short Command Format**

```
·LIBRARY/INSERT LIBRA THIRDRET
```

This operation inserts the object module contained in the file THIRD.OBJ, including all its globals, into the library file LIBRA. Obtain a listing of the library contents, using the LIST option, to verify that the new globals have been added. The listing should look like this:

```
RT-11 LIBRARIAN V05.00 SAT 8-JAN-83 11:05:1
DK:LIBRA.OBJ SAT 8-JAN-83 11:04:21

MODULE      GLOBALS      GLOBALS      GLOBALS
              LEN          PRINT
              ITTOUR
              PUTSTR
```

This listing shows the complete library file containing the globals from all three modules.

You can remove individual globals by using the REMOVE option. For example, to remove the global ITTOUR, type:

LIBRARY/ REMOVE
--------------------

### Long Command Format

```
•LIBRARY/REMOVE(RET)
Library? LIBRA(RET)
Global? ITTOUR(RET)
Global?(RET)
```

### Short Command Format

```
•LIBRARY/REMOVE LIBRA(RET)
Global? ITTOUR(RET)
Global?(RET)
```

The library file's contents now look like this:

```
RT-11 LIBRARIAN V05.00 SAT 8-JAN-83 11:10:22
DK:LIBRA.OBJ          SAT 8-JAN-83 11:10:05

MODULE      GLOBALS      GLOBALS      GLOBALS
            LEN          PRINT
            PUTSTR
```

These are some of the library maintenance operations that you can perform by using the LIBRARY command. Other library operations are available and are explained in the *RT-11 System Utilities Manual*.

#### LIBRARY/MACRO

Create a macro library.

#### LIBRARY/CREATE

Create an object library.

#### LIBRARY/INSERT

Insert object modules into the object library.

#### LIBRARY/LIST[:filespec]

List the current contents of an object library on the line printer; [:filespec] is an optional output file and/or device.

#### LIBRARY/REMOVE

Remove globals from the object library.

**SUMMARY:  
COMMANDS FOR  
MAINTAINING  
LIBRARY FILES**

Since all the object modules used in this chapter already exist as modules within the provided system library SYSLIB, there is no need to save them or the LIBRA library file. You can delete these object modules and their source files from your system

**FILE MAINTENANCE**

volume by using the DELETE command as follows (exclude THIRD.\* from the command line if you did not create this file):

**Long Command Format**

```
.DELETE/NOQUERY(RET)  
Files? FIRST.*,SECOND.*,THIRD.*,LIBRA.OBJ(RET)
```

**Short Command Format**

```
.DELETE/NOQUERY FIRST.*,SECOND.*,THIRD.*,LIBRA.OBJ(RET)
```

FORTRAN IV users who performed the special instructions given in Appendix B should also delete the THIRD files from the storage volume.

**Long Command Format**

```
.DELETE/NOQUERY(RET)  
Files? VOL:THIRD.*(RET)
```

**Short Command Format**

```
.DELETE/NOQUERY VOL:THIRD.*(RET)
```

**REFERENCE**

*RT-11 System Utilities Manual* (AA-M239A-TC). Maynard, Mass.: Digital Equipment Corporation, 1983.

A guide to the use of the RT-11 system utilities.

## CHAPTER 14 DEBUGGING A USER PROGRAM

Debugging is the process of finding and fixing the programming errors that almost every user program initially contains. From your experience in Chapters 9, 10, and 11, you already know about some of the kinds of programming errors that can prevent a program from working properly when you run it on the system.

Frequently, debugging a program requires more time and persistence than writing the program code requires. Therefore, you should anticipate the debugging process throughout the entire program development cycle. That is, you should follow some common programming practices that help you first to make as few programming errors as possible. When errors become apparent during the various phases of development, correct them immediately. Test the validity of any algorithms used within your program. Finally, even though the program appears to be working properly, check it thoroughly with test data.

You can take several steps to decrease the likelihood of introducing errors into your program and to make debugging easier.

First, always use a high-level language if one will suit your programming needs. High-level language programs tend to use fewer statements. English-like words and phrases in the language statements make the program logic easier to follow.

Design the program. The technique of flowcharting the program and then correlating it with the program coding simplifies tracking the program logic and module interrelationships.

Use modular programming. Create the program as a series of smaller, self-contained subprograms. Debug the program in parts.

For frequently used functions, maximize the use of subroutines, subprograms, and — in the case of assembly language programs — macros. These help to structure the program and make it easier to alter or to add features that may be required in the future.

Make use of any software provided by the system, such as library routines and functions. System software has already been debugged and can save you the trouble of re-creating the services.

Make the general flow of a program proceed down the page. Avoid nonstructured branching and convoluted logic, as these

### AVOIDING PROGRAMMING ERRORS

tend to produce programs that are difficult to debug. Finally, use comments liberally throughout the program to show what individual statements or groups of statements do. Use spaces and tabs in the program code to make it easier to read.

Following these preventative steps eliminates many common programming errors and helps to create a programming style. However, even the most careful programmer may overlook a small detail: a typing error during program creation, an undefined label in the code, or some other programming bug. When something is overlooked, debugging becomes necessary.

## **WHEN PROGRAMMING ERRORS OCCUR**

There are three general types of programming errors — syntax, clerical, and logical.

Syntax errors are errors in the physical coding, such as omitting necessary portions of the statement (delimiters for example), reversing the order of information within the statement, or misspelling keywords or instructions.

Clerical errors are non-syntax errors in the physical coding, such as mistyped letters or digits in data. Clerical errors may result in valid statements that do not reflect correct programming logic.

Logical errors are errors made in program development.

The translating program (compiler/assembler/interpreter) generally catches syntax errors and flags them as such in the program listing or on the terminal. On the other hand, you must locate clerical and logical errors by reexamining the program code and logic, using one or more debugging techniques.

Some debugging techniques involve insertion of special debugging code within the program. For example, one way to locate logical errors is to write out intermediate results of a program. You can insert WRITE or PRINT statements at strategic points in the program logic to show the intermediate state of values being calculated. When debugging is complete, you can remove these statements or change them to comments.

You may also find it useful to write a special debugging subroutine that writes out values, particularly if the same variables must be examined in several places or many times.

Another method for finding logic errors — unit testing — is to break the program into smaller parts and test each part separately with artificial data. After you test all parts individually, you can test routine and module linkage — system testing — to see that all related code fits together properly.

Check the program with test data. A standard method for checking out modules is to write a test program that calls the

program with possible options. The test should cause the program to execute all steps in all algorithms. Check programs first with representative data, then with improper data (data that is not in the correct range or size). You should also do volume testing to see that the program works successfully with a representative amount of data.

Each programming language has special debugging aids for examining immediate states. For example, BASIC-11 has a STOP statement that you can insert at strategic points in the program. When the program arrives at a STOP statement, it pauses so that you can use BASIC-11's immediate mode to examine variables, values, and so on. Use an immediate mode GO TO statement pointing to the appropriate line number to continue execution.

FORTTRAN IV has a special DEBUG statement indicator, a D in the first column of a statement line. Operations in statements marked with a D can perform useful debugging functions, such as printing intermediate results. You can treat such statements as source text (and thus execute them) or as comments (and thus ignore them), depending on whether you use a special compiler command option. In addition, FORTTRAN IV has a traceback feature that locates the actual program unit and line number of a run-time error. If the program unit is a subroutine or function subprogram, the error handler traces back to the calling program unit and displays the name of that program unit and the line number where the call occurred. This process continues until the calling sequence has been traced back to a specific line number in the main program unit. Finally, FORTTRAN IV has an optional interactive debugger called FDT (FORTTRAN DEBUGGING TECHNIQUE) that can be linked with a user program.

For MACRO-11 users, RT-11 provides a special on-line debugging tool called ODT (On-line Debugging Technique). This is provided as part of the RT-11 operating system and is an object program on your system volume. It is used exclusively for debugging assembled MACRO-11 programs.

The use of ODT is described next for MACRO-11 users and for those FORTTRAN IV users who will be combining MACRO-11 and FORTTRAN IV program code. Other users can continue to Chapter 15, or go back and perform one of the other language demonstrations. Refer to the reading path outlined in the Preface.

ODT is an interactive debugging tool that allows you to monitor program execution from the console terminal. ODT is provided as the object module ODT.OBJ on your system volume. To use it, you link ODT.OBJ with the assembled MACRO-11

**USING THE ON-LINE  
DEBUGGING  
TECHNIQUE**

## Short Command Format

(Line printer)

(Terminal)

```
,LINK/MAP/DEBUG VOL:SUM␣      ,LINK/MAP:TT:/DEBUG VOL:SUM␣
RT-11 LINK  V08.00      Load Map   Saturday 08-Jan-83 10:06 Page 1
SUM  .SAV      Title: ODT      Ident: V05.00

Section  Addr   Size  Global  Value  Global Value  Global  Value
. ABS.   000000 001000 = 256.  words  (RW,I,GBL,ABS,QVR)
         001000 000372 = 125.  words  (RW,I,LCL,REL,CON)
$ODT$   001372 006152 = 1589. words  (RW,I,LCL,REL,CON)
         0.ODT  001624

Transfer address = 001624, High limit = 007542 = 1969. words
```

Look at the load map, and note that ODT starts at address 1372. The two modules together, ODT and SUM, reside in memory up to location 7542, the high limit. Look at the symbol table listing for the MACRO-11 program. This shows that the program is 372(octal) bytes long and starts at location 1000.

To load and start execution of the load module, use the monitor RUN command. The RUN command brings the entire load module, called SUM.SAV, into the absolute (physical) memory locations shown in the load map and begins execution automatically at the starting, or transfer, address of the first module in memory, which is ODT. Type:

## Long and Short Command Format

```
,RUN SUM␣
ODT V05.00
*
```

ODT prints an identifying message on the terminal and an asterisk indicating that you are in ODT command mode and can enter an ODT command. You are now using ODT to control the execution of your program.<sup>1</sup> ODT commands let you execute the entire program or just portions of it, examine individual locations, examine the contents of the PDP-11 general registers, and change the contents of any locations in your program you wish. If you make a mistake while you are typing any commands, type the DELETE key; ODT responds with a question mark (?) and an asterisk (\*), allowing you to enter another command.

<sup>1</sup>Be sure to read the chapter on ODT in the *RT-11 System Utilities Manual* before you use ODT with any of your own programs. You must observe certain precautions when you write your program and when you load it with ODT. For example, you should make sure that ODT is not loaded into memory locations used by your program. Follow the precautions described in the *RT-11 System Utilities Manual*.

Look at locations 6 through 16 in the assembly listing. With ODT, you can examine these locations in memory as follows (all ODT commands use octal numbers, as does the assembly listing):

```
*1006/012705LF
001010 /000106LF
001012 /012700LF
001014 /000107LF
001016 /012701RET
```

By typing a location address and a slash, you open that location for examination and possible modification. A line feed closes that location and opens the next sequential location for examination. A carriage return simply closes the currently open location.

Note that since the MACRO-11 program was linked to begin at address 1000, you must add the constant 1000 to each address shown in the assembly listing to obtain the actual address used during loading. ODT can do this for you by using special internal locations called relocation registers. Each register can be set to a relocation constant. Thus, if you have linked several modules together, you can set various relocation registers to the corresponding relocation constants of the individual modules. You then indicate in your command which register to use, and ODT automatically adds the constant in that register to the address specified in your command. For example, set relocation register 0 to 1000:

```
*1000;OR
```

Now, to examine locations 0 through 10 in the assembly listing, type:

```
*0,0/012700LF
0,000002 /001342LF
0,000004 /104351LF
0,000006 /012705LF
0,000010 /000106RET
```

In your commands, indicate the number of the relocation register (followed by a comma), since generally you will have more than one register set at a time.

Execute the MACRO-11 program now, using the ODT ;G command, indicating in the command where you wish execution to start. In this case, the program's start (transfer) address is 1000, so type:

```
*0,0;G
THE VALUE OF E IS:
2.5/606/606237.2301314.06525/130440275535025.71477737352744745405502.544
```

As you discovered in Chapter 11, these program results are incorrect. Note that a period has printed, indicating that you are back in monitor command mode. This particular MACRO-11 program returns to the monitor after execution. Therefore, to continue using ODT, you must RUN the load module again:

### Long and Short Command Format

```
.RUN SUM@E
ODT V05.00
*
```

Changes that you make to a program while using ODT, and ODT register assignments that you make, are temporary. Thus, when you restart ODT, you must reenter any commands, such as relocation register commands, that you want to remain in effect. Reset relocation register 0:

```
*1000;0R
```

To help you find programming errors, ODT provides a breakpoint feature. Setting one or more breakpoints in a program causes program control to pause at those locations during execution. When control pauses, ODT prints a short message on the terminal, informing you that a breakpoint has occurred and showing the location at which execution has stopped. This pause returns control to ODT and gives you the opportunity to examine and possibly modify variables or data. Breakpoints are numbered from 0 to 7, so that you can have a total of eight breakpoints set at various instructions in the program at one time.

For example, set breakpoint 0 at location 22 (line 16 in the assembly listing) and breakpoint 1 at location 40 (line 23):

```
*0,22;0B
*0,40;1B
```

Now when you run the program, control pauses first at location 22. Since the breakpoint was set at the instruction at location 22, that instruction has not yet been executed, but all preceding instructions have:

```
*0,0;G
TBO;0,000022
```

Note the message that ODT prints when execution reaches the breakpoint. Normally when execution encounters a breakpoint, only the breakpoint number and location are printed on the

terminal. In this case, the letter T precedes the breakpoint message. This happens because of the way the ODT program uses the console terminal. The assembly instruction .PRINT at line 12 of the assembly listing requests the monitor to print a program message at the same time that ODT needs to print the breakpoint message. ODT, however, has higher priority. By the time the .PRINT request starts to print the program message, execution reaches the breakpoint and gives control to ODT. The .PRINT request has time to print only one character of its message before ODT takes over and prints the breakpoint message. When the program regains control, its message will continue printing from the second character.

Program control has paused at location 22 in the MACRO-11 program. Look in the assembly listing at the instructions that occur there. The instruction at location 16 (line 15) stores the address of the digit vector (at label A) in register 1 (R1). Examine the contents of register 1 to discover what this address is; then open the address and examine its contents and the contents of several addresses following it by using two new ODT commands, \$ and @:

```

*#1/001124 @
0,000124 /000001(LF)
0,000126 /000001(LF)
0,000130 /000001(LF)
0,000132 /000001(RET)

```

The \$ command opens for examination the contents of one of the general PDP-11 registers 0 through 7. The @ command uses the contents of the currently open location as an address and opens that location for examination. Notice that the digit vector A, which begins at location 124, has been initialized to the value 1, the precise value indicated by the comments at line 48 of the program listing.

If you were to continue program execution now, the branch instruction at line 22 of the assembly listing would cause program control to loop back to the instruction at line 16 where breakpoint 0 is set, again causing execution to pause. Since you wanted to continue to the next breakpoint (set at location 40), you must first cancel the breakpoint at location 22. To do this, type:

```
*#0B
```

This removes the breakpoint at location 22. The number (in this case 0) indicates which breakpoint is to be removed. Now continue program execution using the ;P command (proceed from breakpoint). Execution progresses through the loop and continues until it reaches the breakpoint set at location 40:

```
*;P  
HB1;0,000040
```

(Note that the monitor has time to print the second character, and perhaps additional characters, of the program message before ODT gains control.) Now examine the contents of several of the program vector locations beginning at location 124:

```
*0,124/000012Ⓞ  
0,000126 /000012Ⓞ  
0,000130 /000012Ⓞ  
0,000132 /000012Ⓞ
```

The instructions prior to the breakpoint at location 40 constitute a multiplication routine. This routine multiplies the vector contents by 10 (12 octal), as you have just verified.

You can see how the breakpoint feature is a very useful debugging aid. It allows you to execute selected portions of a program and verify that data and variables are being used correctly during execution. You can use the breakpoint feature to locate the error that is in this program.

First, clear all previously set breakpoints (in this case, there is only the one at location 40) by typing the ;B command with no argument.

```
*;B
```

Now set a breakpoint at location 110 (line 41 of the assembly listing). You want to verify the data that is being passed to the monitor in register 0 in the ADD instruction in line 40.

```
*0,110;0B  
*;P  
EB0;0,000110
```

Now examine the contents of register 0.

```
**0/000065 \065 =5Ⓞ
```

At this point in execution, register 0 contains 000065. The backslash (\) command prints the low-order byte of the opened location on the console terminal and also converts this to an ASCII character (if it is a valid ASCII code) and prints the character. In this case, the number 5 prints. If you look back at the program results printed earlier in this chapter, you can see that 5 is the first digit of the tabulated result (following the message *THE VALUE OF E IS 2*). If you are experienced in mathematics, you know this result is incorrect because the approximate value of E is 2.718. And you now also know that the

program error is not in the interface to the monitor service used to print the result (.TTYOUT), but that it occurs somewhere before location 110. So the next step in debugging this program is to set a breakpoint at some earlier point in the program logic and to rerun the program. You must restart ODT to do this. Return to monitor mode by typing CTRL/C. The remainder of the program message prints on the terminal; then the monitor period appears, indicating that you are in monitor mode.

```
*CTRL/C
VALUE OF E IS:
Z.
.
```

Restart ODT and reset relocation register 0.

```
.RUN SUM(RET)
ODT  V05.00
*1000;OR
```

Set a breakpoint at location 76 (line 37 in the assembly listing), and start program execution at its beginning.

```
*0,76;OB
*0,0;G
TB0;0,000076
```

Again, examine register 0 to verify its contents.

```
**0/000033(RET)
```

By following the program logic in the assembly listing, you know that the value in register 0 should at this point be 33(octal) (2.7, previously multiplied by 10, = 27[decimal] = 33[octal]). So the value in register 0 is correct. From this, you can deduce that the error must occur somewhere between locations 76 and 110. The proper step now is to check the assembly listing, where you find the error at line 40. The decimal point that should follow the 10, identifying it as a decimal 10, is missing. Therefore the program treats the 10 as an octal 10, or 8(decimal), making each digit in the result off by an additive factor of 2. The data in location 106, then, should be 72, not 70. Since this data has not yet been used, you can change it now with ODT and continue program execution; if it had been used, you would need to restart ODT and then change the data. To change the contents of a location, simply open the location, type in the new contents, and close the location, using a carriage return.

```
*0,106/000070 72(RET)
```

Now eliminate all breakpoints.

```
*;B
```

Continue program execution; the correct results should print.

```
*;P
THE VALUE OF E IS:
2.7182818284590452353602874713526624977572470936999595749669676277240766
```

**SUMMARY:  
COMMANDS FOR  
DEBUGGING  
PROGRAMS**

To Start ODT

**LINK/DEBUG**

Link the assembled program (the program to be debugged) with the ODT object module.

To Use ODT<sup>1</sup>

**(LF)**

Close the currently open location and open the next sequential location for examination and possible modification.

**(RET)**

Close the currently open location.

**addr/**

Open the location indicated (addr) for examination and possible modification.

**addr;G**

Begin program execution at the indicated address (addr).

**;P**

Continue program execution from a previous breakpoint.

**addr;nB**

Set one of the eight available breakpoints (n) at the indicated address (addr).

**;nB**

Cancel the indicated breakpoint (n).

**;B**

Cancel all breakpoints.

**addr;nR**

Set one of the eight available relocation registers (n) to the relocation constant value indicated by addr.

<sup>1</sup>Only a very few of the available debugging commands have been demonstrated in this chapter. Consult the *RT-11 System Utilities Manual* for all ODT commands.

\$n

Open one of the eight general registers (n) for examination and possible modification.

@

Use the contents of the currently open location as an address; close the currently open location; go to the new address, and open it for examination and possible modification.

\  
Print on the console terminal the low-order byte of the currently open location; if possible, convert the value to an ASCII code and print the corresponding character on the terminal.

Changes you make with ODT are temporary. Therefore you should now use the editor to correct the source program SUM.MAC. You should edit line 40 so that it reads:

## FILE MAINTENANCE

```
ADD      #10.+ '0,RO          ;MAKE DIGIT ASCII
```

The file SUM.MAC is currently stored on the storage volume VOL:. Edit this file, then reassemble, relink, and rerun it to verify that it is correct. When you have done this, store the updated version of the source file on the storage volume under the same name (SUM.MAC), including the files SUM.OBJ and SUM.SAV.

After you have corrected and rerun the program, continue on to Chapter 15, or go back and perform one of the other language demonstrations. Refer to the reading path outlined in the Preface.

*RT-11 System Utilities Manual* (AA-M239A-TC). Maynard, Mass.: Digital Equipment Corporation, 1983.

## REFERENCE

A guide to the use of the RT-11 system utilities.



## CHAPTER 15 USING THE FOREGROUND/BACKGROUND MONITOR

A special feature of the RT-11 operating system is that it provides a choice of operating environments. You are using its foreground/background environment. This environment allows two independent programs to reside in memory at the same time and to execute concurrently.

You have used the foreground/background (FB) monitor to control the system and to perform the various exercises in this manual.

The foreground/background environment is designed so that two programs can — but need not — share memory and run concurrently. One of these programs you designate as the foreground program. The system gives priority to the foreground program (or job, as it is usually called) and allows it to run until some condition, perhaps waiting for an I/O completion, causes it to relinquish control to the other program (the background job). The system then allows the background job to run until the foreground job again requires control, and so on. In this way, the two programs share system resources. Whenever the foreground program is idle, the background program runs. Yet whenever the foreground program requires service, its requests are immediately satisfied. To the user at the terminal, the two programs appear to run simultaneously.

Foreground priority programs are generally time-critical. For example, you may want to designate as the foreground job a program that collects and analyzes data. Background programs are usually non-time-critical. Thus, you can continue to do program development as the background job by using monitor commands to run the editor, the FORTRAN IV compiler, the linker, and so forth.

In order to perform the following exercises your system must have a clock. Verify whether your system has a clock by entering the TIME command twice. If the time displayed changes, your system has a clock. If your system does not have a clock you should skip ahead to Chapter 16.

Two programs are provided for you to run a foreground/background demonstration. These programs reside on your system volume. The background job is called DEMOBG, and the foreground job DEMOFG. The function of the foreground job is to send messages every two seconds to the background job, telling it to ring the terminal bell. The background job recognizes

### THE FOREGROUND/ BACKGROUND ENVIRONMENT

### RUNNING THE FOREGROUND/ BACKGROUND PROGRAMS

these messages and rings the bell once for each message sent by the foreground job.

Although the foreground job is always active, sending messages to the background job every two seconds, other programs besides DEMOBG can be executed in the background. Only when DEMOBG is active, however, is the circuit complete so that messages can be successfully received and honored. During the periods when DEMOBG is not running, the foreground program enters the messages in the monitor message queue. Once you restart DEMOBG in the background, the system immediately dequeues all the messages since the last exit of DEMOBG, resulting in many successive bell rings. When the queue is empty, the normal send/receive cycle resumes, and the bell rings every two seconds as each current message is sent and honored.

### **Creating the Background Job**

The background program DEMOBG.MAC is an assembly language source file and must be assembled and linked before you can use it. When you execute DEMOBG, it displays a message on the terminal. It is assumed that you have set the date.

### **Editing the Background Job**

Use the text editor to modify the background job, DEMOBG.MAC. One of the lines of the message that is output by the program has a semicolon character preceding it, which makes the line a comment field. This will prevent the line from being printed as part of the message. Thus, the semicolon must be deleted from that line.

Change the line

```
; .ASCII /WELL DONE./
```

to

```
.ASCII /WELL DONE./
```

### **Running the Background Job**

If you performed the demonstration in Chapter 11, you are already familiar with assembly/link operations, and the following command explanations can serve as a review. If you did not read Chapter 11, simply type the command lines as shown.

Assemble the background job.

### **Long Command Format**

```
,MACRO(RET)  
Files? DEMOBG/LIST(RET)
```

### Short Command Format

```
.MACRO DEMOBG/LIST(RET)
```

Link the .OBJ file produced by the assembler to create a runnable job.

### Long Command Format

```
.LINK(RET)  
Files? DEMOBG(RET)
```

### Short Command Format

```
.LINK DEMOBG(RET)
```

Now run the background job and check the results.

```
.RUN DEMOBG(RET)  
RT-11 DEMONSTRATION PROGRAM  
IF INCORRECTLY EDITED, THIS IS THE LAST LINE.  
WELL DONE.
```

If you did not delete the semicolon character, the last line will not be output. Return to the monitor by typing two successive CTRL/Cs.

```
(CTRL/C)
```

```
(CTRL/C)
```

```
^C  
^C
```

The FB monitor provides you with commands that allow you to control the two-job environment. They let you interact with the two jobs and let the two jobs interact with one another.

## USING THE FB MONITOR

When two jobs run simultaneously, you must have some means of indicating the job to which you are directing commands. Likewise, the two jobs must have the means to identify themselves when they have messages to print. The following are some conventions that apply to system communication in a two-job environment.

### Communication in a Two-Job Environment

1. The foreground job has priority. If both the foreground and the background job are ready to print output at the same time, the foreground job prints first. The foreground job prints a complete line, then the background job prints a complete line, and so on.

2. Either job can interrupt your input at the terminal if it has a message to print.
3. Messages printed by the background job are preceded by the characters B>.
4. Messages printed by the foreground job are preceded by the characters F>.
5. Typed commands are initially directed to the background job. You can redirect control alternately to the foreground and background jobs by using the CTRL/F and CTRL/B commands.

To direct typed input to the foreground job, type CTRL/F. This command instructs the monitor that all subsequent terminal input — commands and text — is directed to the foreground job. Typing this command causes the system to print an F> on the terminal, unless output is already coming from the foreground job. Command input remains directed to the foreground job until the foreground job terminates, or until it is redirected to the background job through CTRL/B.

To direct typed input to the background job, type CTRL/B. This command instructs the monitor that all subsequent terminal input — commands and text — is directed to the background job. Typing this command causes the system to print a B> on the terminal, unless output is already coming from the background job. Command input remains directed to the background job until redirected to the foreground job through CTRL/F.

## Creating the Foreground Job

The foreground program DEMOFG is an assembly language source file; it must be assembled and linked before you can use it.

### Long Command Format

```
,MACRORET  
Files? DEMOFG/LISTRET
```

### Short Command Format

```
,MACRO DEMOFG/LISTRET
```

LINK/ FOREGROUND
---------------------

The output resulting from this MACRO command includes an object file called DEMOFG.OBJ and a listing file called DEMOFG.LST. The command creates both files on your system volume. You must link the .OBJ file to produce a runnable foreground program. You use the LINK command, just as you have in earlier chapters, but you also use the /FOREGROUND

option.<sup>1</sup> This option produces a load module with a .REL file type which signifies to the system that the file is a foreground program and is to be run as the priority job.

### Long Command Format

```
.LINK/FOREGROUND(RET)  
Files? DEMOFG(RET)
```

### Short Command Format

```
.LINK/FOREGROUND DEMOFG(RET)
```

Now you are ready to operate the two-job environment. Many times, you have to consider the devices that are used for output in a foreground/background environment. For example, if your program assumes that the output device is a line printer, and you do not have a line printer or you want to output to another device, use the ASSIGN command. Type this command in the following way, substituting the two-letter mnemonic from Table 4-2 for the storage volume in place of dd.

## Executing the Foreground and Background Jobs

### Long Command Format

```
.ASSIGN(RET)  
Physical device name? dd:(RET)  
Logical device name? LP:(RET)
```

### Short Command Format

```
.ASSIGN dd: LP:(RET)
```

You do not have to consider the above information for the demonstration programs that are provided, since the foreground job communicates with the background job, and both jobs send their output to the terminal.

When you use the FB monitor, you must always load into memory the peripheral device handlers needed by the foreground job. You use the monitor LOAD command to make a device handler permanently resident in memory. For example, if your foreground job requires the use of the line printer, you must load the LP device handler. You must specify the jobtype with the command. For a foreground job, the jobtype is F; for a background job, the jobtype is B. If you have assigned the code LP: to another device, the system automatically loads the assigned handler and you need not enter a LOAD command. If you are using the line printer, type:

LOAD

<sup>1</sup>This command option also applies to compiled FORTRAN IV programs that are to be linked as a foreground job.

### Long Command Format

```
,LOAD␣  
Device? LP:=F␣
```

### Short Command Format

```
,LOAD LP:=F␣
```

FRUN

The command to load and start execution of the foreground job is FRUN, which is similar to the RUN command except that the system automatically loads and starts the execution of the foreground .REL program. (To execute a FORTRAN IV foreground job, you must use the /BUFFER:n option with the FRUN command. The argument n represents, in octal, the number of words of memory to allocate.) Use this command to start the execution of DEMOFG.REL.

### Long and Short Command Format

```
,FRUN DEMOFG␣
```

```
F>  
FOREGROUND DEMONSTRATION PROGRAM  
SENDS A MESSAGE TO THE BACKGROUND PROGRAM "DEMOBG"  
EVERY 2 SECONDS, TELLING IT TO RING THE BELL.  
B>
```

The foreground program DEMOFG is now running and queuing the message for the background program every two seconds. You now execute the background program DEMOBG to allow it to receive the messages that were queued and to ring the bell.

```
,RUN DEMOBG␣  
RT-11 DEMONSTRATION PROGRAM  
IF INCORRECTLY EDITED, THIS IS THE LAST LINE.  
WELL DONE.
```

The bell rings several times in rapid succession as the monitor dequeues the messages, and then every two seconds as the foreground job sends its message to the background job.

You can run other jobs in the background. First, terminate the background job DEMOBG, using the double CTRL/C command.

```
␣  
␣
```

Execute a DIRECTORY command in the background to get a listing of all the .OBJ files on the system volume by typing:

```
,DIRECTORY *.OBJ␣
```

The foreground job is still running and queuing its messages to the monitor. Rerun the background program to collect all the foreground messages while the background job was stopped and the directory was printing.

```
. RUN DEMOBC(RET)
RT-11 DEMONSTRATION PROGRAM
IF INCORRECTLY EDITED, THIS IS THE LAST LINE.
WELL DONE.
```

The bell again rings several times in succession and then rings once every two seconds. Stop the background job by using the double CTRL/C command.

```
CTRL/C
CTRL/C
```

Now stop the foreground job and remove it from memory. To do this, you must first use the CTRL/F command to direct terminal input to the foreground. Type:

```
. CTRL/F
F>
```

The system prints the characters F> to remind you that you are now directing command input to the foreground job. Use the double CTRL/C command to interrupt and terminate the execution of the foreground job, and return control to the background job.

```
CTRL/C
CTRL/C
B>
```

You should unload the foreground job to reclaim memory space for background use. Use the monitor UNLOAD command as follows:

UNLOAD

### Long and Short Command Format

```
UNLOAD F(RET)
```

F represents the foreground job; you should use this code whenever you want to unload the foreground job. To unload any loaded device handlers, you must use their two-letter device mnemonics.

Check to see if the .LST files were produced as a result of this demonstration.

## Long and Short Command Format

```
. DIRECTORY *.LST
  08-Jan-83
DEMOBG.LST      4  08-Jan-83  DEMOFG.LST      6  08-Jan-83
  2 Files, 10 Blocks
  406 Free blocks
```

The foreground program has access to all the system features available to a background program — opening and closing files, reading and writing data, and so on. However, before you begin to write and use programs in the foreground, read the *RT-11 Software Support Manual* for coding restrictions.

### SUMMARY: COMMANDS USED IN AN FB ENVIRONMENT

#### CTRL/B

Direct all keyboard input to the background job (until CTRL/F).

#### CTRL/F

Direct all keyboard input to the foreground job (until CTRL/B).

#### FRUN

Load and start execution of the foreground job.

#### LOAD dd

Bring the indicated device handler into memory; the handler becomes resident in memory.

#### UNLOAD dd

Take the indicated device handler out of memory, reclaiming its memory space; the handler becomes nonresident in memory.

#### UNLOAD F

Reclaim the memory space used by the foreground job.

### FILE MAINTENANCE

You assembled the source file DEMOFG.MAC and produced an .OBJ file, linking it to produce DEMOFG.REL. You also created a .LST file named DEMOFG.LST on your system volume. You should save on your storage volume the files DEMOFG.REL and DEMOFG.MAC, and delete from your system volume the files DEMOFG.OBJ and DEMOFG.LST. Do not delete DEMOFG.MAC, since this file was distributed as part of the RT-11 operating system. Do the same for the file DEMOFG, which you created as a .SAV file instead of a .REL file.

## Long Command Format

```
.COPY(RET)
From? DEMOFG.MAC,DEMOFG.REL(RET)
To ? VOL:*,*(RET)
Files copied:
DK:DEMOFG.MAC    to VOL:DEMOFG.MAC
DK:DEMOFG.REL   to VOL:DEMOFG.REL
```

```
.DELETE(RET)
Files? DEMOFG.OBJ,DEMOFG.LST(RET)
```

## Short Command Format

```
.COPY DEMOFG.MAC,DEMOFG.REL VOL:*,*(RET)
Files copied:
DK:DEMOFG.MAC    to VOL:DEMOFG.MAC
DK:DEMOFG.REL   to VOL:DEMOFG.REL

.DELETE DEMOFG.OBJ,DEMOFG.LST(RET)
```

Finally, obtain a brief directory listing of your storage volume so that you can see its current status:

## Long and Short Command Format

```
.DIRECTORY/BRIEF VOL:(RET)
```

*RT-11 Software Support Manual* (AA-H379B-TC). Maynard, Mass.: Digital Equipment Corporation, 1983.

A technical manual providing RT-11 programming concepts.

*RT-11 System User's Guide* (AA-5279C-TC). Maynard, Mass.: Digital Equipment Corporation, 1983.

A guide to the use of the RT-11 operating system.

## REFERENCES



## CHAPTER 16 USING INDIRECT FILES

The RT-11 system provides an operational aid called an indirect file, which allows the system to run unattended. An indirect file is a file composed entirely of monitor operating commands. When you start the execution of the indirect file, the monitor processes these commands in consecutive order. So once you have created an indirect file and started its execution, you can direct your attention to other tasks or even physically leave the system, since the monitor executes the commands automatically and consecutively.<sup>1</sup>

The kinds of operations that RT-11 can best perform in an indirect file are those that involve much computer processing but that do not require your supervision or intervention. For example, multiple assemblies, compilations, and data transfer operations are ideal operations for indirect file processing. Also, any series of commands that you are likely to type often can easily run as an indirect file.

Use the editor to create an indirect file as a text file. You can call the file by any file name you wish, but you should give it a file type of .COM, since this file type is the default used by the monitor to locate the file.

You structure the lines of text that make up an indirect file just like keyboard input. Thus, if you were to list the indirect file it would look like terminal keyboard text without any monitor prompts.

You enter monitor commands into the indirect file as you would on the terminal. As an example, both of the following accomplish the same operation when executed as part of an indirect file:

```
COPY@  
INFIL.MAC@  
OUTFIL.MAC@  
  
COPY INFIL.MAC OUTFIL.MAC@
```

Since monitor prompts are not included in the indirect file, using the long command format requires that you anticipate

<sup>1</sup>The indirect file concept is similar to BATCH processing. Although indirect files lack many of the BATCH capabilities, they are easier to use than BATCH. (The RT-11 computer system also supports a BATCH processor, as described in the *RT-11 System Utilities Manual*.)

### CREATING AN INDIRECT FILE

### Entering Monitor Commands

each prompt and its proper response. It is suggested that you use the short command format and enter the command as a single line of text. Terminate each command line with a carriage return.

## Using the Editor to Create an Indirect File

The indirect file that you will now create incorporates several of the commands previously demonstrated in this manual. Thus it serves both as an example of the format of indirect file input and as a brief review of the monitor commands used to copy, process, and delete files. In addition, one new command, DEASSIGN, is demonstrated.

List a directory of your storage volume. The only files that should appear in the listing are GRAPH.FOR, SUM.MAC, and MATCH.BAS. All three files will be in the directory only if you performed the exercises for all three languages (FORTRAN IV, MACRO-11, and BASIC-11).

### Long and Short Command Format

```
.DIRECTORY VOL:␣  
  08-JAN-83  
SUM  .MAC   3  08-JAN-83      MATCH .BAS   3  08-JAN-83  
GRAPH .FOR   2  08-JAN-83  
  3 Files, 8 Blocks  
  4754 Free blocks
```

If any other files are listed, delete them using the DELETE command before you create the indirect file.

Use the EDIT/CREATE monitor command to create a file called INDCT.COM, inserting the commands according to the directions in the right-hand column. When you have finished creating the file, list it and check for typing errors. Correct any errors you find, and then close the file, using the EX editing command.

### Long and Short Command Format

```
.EDIT/CREATE INDCT.COM␣  
*IDATE 8-JAN-83␣  
TIME 8:00:00␣
```

Enter a hypothetical date and time (if your system has a clock).

```
DATE␣
```

Print the date.

```
DEASSIGN␣
```

Deassign all previous device assignments and set new ones as follows:

```
ASSIGN TT: LP:␣
```

Assign the logical name LP: to the terminal.

```
ASSIGN ddn VOL:␣
```

Assign the logical name VOL: to the storage volume (dd).

DIRECTORY/BRIEF VOL:␣	List an abbreviated directory of VOL:.
COPY VOL:GRAPH,FOR GRAPH,FOR␣	FORTRAN IV users insert this command to copy the FORTRAN IV demo program to the system volume.
COPY VOL:SUM,MAC SUM,MAC␣	MACRO-11 users insert this command to copy the MACRO-11 demo program to the system volume.
COPY VOL:MATCH,BAS MATCH,BAS␣	BASIC-11 users insert this command to copy the BASIC-11 demo program to the system volume.
FORTRAN/LIST GRAPH␣ LINK/MAP GRAPH,SYSLIB,FORLIB␣	FORTRAN IV users who do not need to load the language volume include these commands to compile and link the FORTRAN IV demo program.
MACRO/LIST/CROSSREFERENCE SUM␣ LINK/MAP SUM␣	All users assemble and link the demo program.
RENAME MATCH,BAS MATCH,MAP␣	BASIC-11 users simply rename the demo program.
MACRO/LIST/CROSSREFERENCE DEMOFG␣ LINK/BACKGROUND/MAP DEMOFG␣	All users assemble and link the DEMOFG file.
DIRECTORY *,OBJ␣	List a directory of object files.
DELETE/NOQUERY GRAPH,*␣	FORTRAN IV users delete the GRAPH files.
DELETE/NOQUERY SUM,*␣	MACRO-11 users delete the SUM files.
DELETE MATCH,MAP␣	BASIC-11 users delete the MATCH file.
DEASSIGN␣	Deassign all device assignments.
TIME␣	If your system has a clock, print the time to show how long total processing took.
␣␣ *B/L␣␣ DATE 8-JAN-83 TIME 8:00:00	Terminate the insert command and list the indirect file to check for errors. (Example input is shown here.)

```
DATE
DEASSIGN
ASSIGN TT: LP:
ASSIGN RK1: VOL:
DIRECTORY/BRIEF VOL:
COPY VOL:GRAPH.FOR GRAPH.FOR
COPY VOL:SUM.MAC SUM.MAC
COPY VOL:MATCH.BAS MATCH.BAS
FORTRAN/LIST GRAPH
LINK/MAP GRAPH,SYSLIB,FORLIB
MACRO/LIST/CROSSREFERENCE SUM
LINK/MAP SUM
RENAME MATCH.BAS MATCH.MAP
MACRO/LIST/CROSSREFERENCE DEMOFG
LINK/BACKGROUND/MAP DEMOFG
DIRECTORY *.OBJ
DELETE/NOQUERY GRAPH.*
DELETE/NOQUERY SUM.*
DELETE MATCH.MAP
DEASSIGN
TIME
*EXESCESC
```

Close the file INDCT.COM.

## EXECUTING AN INDIRECT FILE

Once you have created an indirect file and checked it for errors, you are ready to start its execution. The command to start execution of an indirect file is the at sign (@) followed by the appropriate file name (the file type .COM is assumed unless you indicate otherwise). Execution starts immediately, and the system processes commands in the indirect file in consecutive order. Each command is echoed on the terminal as it is processed. If an error within the indirect file affects the processing of a command, the system prints a system message on the terminal and stops execution of the entire file. Therefore, it is particularly important that you check your indirect file for errors before you start it and then leave the area. You can stop execution of an indirect file at any time by typing two CTRL/Cs.

Run the indirect file that you have just created by typing:

```
._@INDCTRET
```

It takes a minute or two for the commands in this file to be processed and for the listings to print. If your system has a clock, the time printed at the end of execution tells you exactly how long command processing has taken. Following is an example run.

```
._@INDCT
.DATE 8-JAN-83
.TIME 8:00:00
.DATE
8-Jan-83
.DEASSIGN
.ASSIGN TT: LP:
.ASSIGN RK1: VOL:
.DIRECTORY/BRIEF VOL:
08-Jan-83
GRAPH.FOR SUM.MAC MATCH.BAS
3 Files, 8 Blocks
4754 Free Blocks
.COPY VOL:GRAPH.FOR GRAPH.FOR
.COPY VOL:SUM.MAC SUM.MAC
```

.COPY VOL:HATCH.BAS HATCH.BAS
FORTRAN/LIST GRAPH
FORTRAN IV V02.5 Sat 08-Jan-83 08:00:16 PAGE 001

```
C GRAPH.FOR VERSION 1
C THIS PROGRAM PRODUCES A PLOT ON THE TERMINAL
C OF AN EXTERNAL FUNCTION, FUN(X,Y)
C THE LIMITS OF THE PLOT ARE DETERMINED BY THE DATA STATEMENTS
C 'STAB' IS FILLED WITH A TABLE OF HEIGHT FLAGS
C 'STRING' IS USED TO BUILD A LINE OF GRAPH FOR PRINTING
0001 SCAL(ZMIN,ZMAX,K)*ZMINH*(DAT(K-1)*(ZMAX-ZMIN)/FLOAT(MAXZ-1)
0002 LDDICL#1 STRING(133),STAB(100)
0003 DATA XMIN,XMAX,MAXX/-5,0,5,0,45/
0004 DATA YMIN,YMAX,HMAXY/-5,0,5,0,72/
0005 DATA FMAX,0,0,1,0/
0006 CALL SCOPY(' 1 2 3 4 5 6 7 8 9 '+,STAB)
0007 MAXF=LEN(STAB)
0008 DD 20 IX=1,MAXX
0009 Y=SCAL(YMIN,YMAX,HMAXY,IX)
0010 CALL REPEAT('*',STRING,HMAXY)
0011 IF(IX.EQ.1 .OR. IX.EQ.MAXX) GO TO 20
0013 DD 10 IY=2,HMAXY-1
0014 Y=SCAL(YMIN,YMAX,HMAXY,IY)
0015 IFUN=2*INT(FLOAT(MAXF-3)*F(X,Y)-FMIN)/(FMAX-FMIN)
0016 10 STRING(IY)=STAB(MIN(MAXF,HMAXO(1),IFUN))
0017 20 CALL PUTSTR(7,STRING,' ')
0018 CALL EXIT
0019 END
```

.HAIN.
?FORTRAN-I-E.HAIN.J Errors: 0, Warnings: 2
FORTRAN IV Storage Map for Program Unit .HAIN.

Local Variables: .PSECT \$DATA, Size = 000470 ( 156 words)

Table with 8 columns: Name, Type, Offset, Name, Type, Offset, Name, Type, Offset. Lists variables like FMAX, IX, MAXF, MAXZ, XMIN, YMIN, XMAX, YMAX, ZMIN, ZMAX.

Local and COMMON Arrays:

Table with 5 columns: Name, Type, Section, Offset, Size, Dimensions. Lists arrays STAB and STRING.

Subroutines, Functions, Statement and Processor-Defined Functions:

Table with 12 columns: Name, Type, Name, Type, Name, Type, Name, Type, Name, Type, Name, Type. Lists subroutines EXIT, MAXD, XMIN, YMIN, SCOPY, FORTRAN IV.

FORTRAN IV V02.5 Sat 08-Jan-83 08:01:46 PAGE 001

```
0001 FUNCTION FUN(X,Y)
0002 R=SQRT(X**2+Y**2)
0003 FUN=(X*Y*R*EXP(-R))**2
0004 RETURN
0005 END
FUN
```

FORTRAN IV Storage Map for Program Unit FUN

Local Variables: .PSECT \$DATA, Size = 000024 ( 10 words)

Table with 8 columns: Name, Type, Offset, Name, Type, Offset, Name, Type, Offset. Lists variables FUN, Y.

Subroutines, Functions, Statement and Processor-Defined Functions:

Table with 12 columns: Name, Type, Name, Type, Name, Type, Name, Type, Name, Type, Name, Type. Lists subroutines EXP.

.LINK/MAP GRAPH:SYSLIB:FORLIB

RT-11 LINK V08.00 Load Map Saturday 08-Jan-83 08:02 Page 1
GRAPH -SAV Title: .HAIN. Ident: FORV02

Section Addr Size Global Value Global Value Global Value

Large table listing memory sections and their values. Includes sections like ABS, DTS#1, STK#1, etc. with columns for address, size, and global values.

RT-11 LINK V08.00 Load Map Saturday 08-Jan-83 08:02 Page 2
GRAPH -SAV Title: .HAIN. Ident: FORV02

Table listing memory sections and their values, continuing from the previous table. Includes sections like MOI#HP, MOI#PA, ISN#, etc.

# Using Indirect Files

```

LLE# 007506 LEB# 007510 LBT# 007516
LDE# 007520 LNE# 007530 LLT# 007532
TSL#S 007536 TSL#M 007542 TSL#I 007546
TSL#P 007554 HAXO 007562 HINO 007604
RET#L 007632 RET#F 007656 RET#I 007644
RET# 007644 #OTIS 007702 #TOTIS 007704
TUL# 010024 #TUL 010024 TUF# 010032
#TUF 010032 TVD# 010040 #TVD 010040
TVB# 010044 #TVD 010046 TVP# 010054
#TVP 010054 TVI# 010062 #TVI 010062
SAL#IH 010216 SAL#SH 010220 SUL#IH 010224
SUL#SH 010226 SAL#SHM 010234 SUL#SHM 010240
SAL#IP 010244 SAL#SP 010246 SUL#IP 010252
SUL#SP 010254 SAL#MP 010262 SUL#MP 010266
#CUTFB 010272 #CUTFI 010272 #CUTCB 010306
#CUTCI 010306 #CUTDB 010306 #CUTDI 010306
CIC# 010320 CID# 010320 CLC# 010320
CLB# 010320 #BI 010320 CIP# 010330
CLF# 010330 #I 010330 CIL# 010442
CLI# 010446 #INITI 010450 #CLOSE 010546
#ERRTB 011344 #ERRS 011451 #FCHNL 015212
#FI 016054 #FID 016060 #PUTRE 017224
#PUTBL 017332 #ETBL 017742 #EDEFIL 020126
#EDF2 020142 #AVRG# 020162 THRD# 020340
#STPS 020342 STP# 020350 #STP 020350
FOD# 020354 #EXIT 020374 #WAIT 020520
#VPRINT 020562 #DUMPL 020574

DTS#P 020722 000054 = 22. words (RW,D:GBL,REL,DVR)
SYS#I 020776 000244 = 82. words (RW,I:LCL,REL,CON)
USER#I 021242 000000 = 0. words (RW,I:LCL,REL,CON)
#CODE 021242 001316 = 359. words (RW,I:LCL,REL,CON)
DTS#D 022560 001036 = 271. words (RW,I:LCL,REL,CON)
SYS#D 023616 000000 = 0. words (RW,I:LCL,REL,CON)
#DATAP 023616 000106 = 35. words (RW,D:LCL,REL,CON)
DTS#D 023724 000006 = 3. words (RW,D:LCL,REL,CON)
DTS#S 023732 000002 = 1. words (RW,D:LCL,REL,CON)
#ADTS 023732
SYS#S 023734 000004 = 2. words (RW,D:LCL,REL,CON)
#SYS#LB 023734 #LOCK 023736 #CRASH 023737
USER#D 024476 000000 = 0. words (RW,D:LCL,REL,CON)
****. 024476 000000 = 0. words (RW,D:GBL,REL,DVR)
RT-11 LNK# V05.00 Load Map Saturday 08-Jan-83 08:02 Page 3
GRAPH ,SAV Title: .MAIN. Ident: FORV02

```

Transfer address = 021242, High limit = 024474 = 527B. words

.MACRO/LIST/CROSSREFERENCE SUM  
SUM.MAC VERSION 1 MACRO V05.00 Saturday 08-Jan-83 08:07 Page 1

```

1 .TITLE SUM.MAC VERSION 1
2
3 .MCALL ,TTYOUT, .EXIT, .PRINT
4
5
6
7 000106 N = 70. #NO. OF DIGITS OF 'E' TO CALCULATE
8 # 'E' = THE SUM OF THE RECIPROCAL OF THE FACTORIALS
9 1/0! + 1/1! + 1/2! + 1/3! + 1/4! + 1/5! + ...
10
11 000000 EXP: .PRINT #MESSAD #PRINT INTRODUCTORY TEXT
12 000006 012705 000104 MOV #N,R5 #NO. OF CHARS OF 'E' TO PRINT
13 000012 012700 000107 FIRST: MOV #N,I,R0 #NO. OF DIGITS OF ACCURACY
14 000016 012701 000124 #A,R1 #ADDRESS OF DIGIT VECTOR
15 000022 004311 SECONDD: ASL R1 #DO MULTIPLY BY 10 (DECIMAL)
16 000024 011146 MOV R1,(-SP) #SAVE #2
17 000026 004311 ASL R1 #4
18 000030 004311 ASL R1 #8
19 000032 004321 DEC #SP)+,(R1)+ #NO. #10, POINT TO NEXT DIGIT
20 000034 005300 BNE SECONDD #BRANCH IF NOT
21 000036 001371 MOV #N,R0 #DO THRU ALL PLACES, DIVIDING
22 000040 012700 000106 THIRDD: MOV #N,I,R3 #BY THE PLACES INDEX
23 000044 014103 MOV #(-R1),R2 #INIT QUOTIENT REGISTER
24 000046 012702 177777 FOURTH: INC R2 #BUMP QUOTIENT
25 000052 005202 SUB R0,R3 #SUBTRACT LOOP ISN'T BAD
26 #NUMERATOR IS ALWAYS < 10#N
27 000054 160003 BCC FOURTH #FIX REMAINDER
28 000056 103375 ADD R0,R3 #SAVE REMAINDER AS BASIS
29 000060 006003 MOV R3,R1 #FOR NEXT DIGIT
30 000062 010311 ADD R2,-2(R1) #GREATEST INTEGER CARRIES
31 #TO GIVE DIGIT
32 000064 060261 177776 DEC R0 #AT END OF DIGIT VECTOR
33 #BRANCH IF NOT
34 000070 005300 BNE THIRDD #GET DIGIT TO OUTPUT
35 000072 001364 MOV -(R1),R0 #FIX THE 2,7 TO .7 SO
36 000074 014100 FIFTH: SUB #10.,R0 #THAT IT IS ONLY 1 DIGIT
37 000076 162700 000012 BCC FIFTH #REALLY DIVIDE BY 10)
38 #MAKE DIGIT ASCII
39 000102 103375 ADD #10.,R0 #OUTPUT THE DIGIT
40 000104 062700 000072 .TTYOUT #CLEAR NEXT DIGIT LOCATION
41 000110 CLR R1 #MORE DIGITS TO PRINT?
42 000114 005011 DEC R5 #BRANCH IF YES
43 000116 002305 BNE FIRST #WE ARE DONE
44 000120 001334 .EXIT
45 000122
46
47 000124 000107 A: .REPT N+1
48 .WORD 1 #INIT VECTOR TO ALL ONES
49 .ENDD
50
51 000342 124 110 105 MESSAB: .ASCII /THE VALUE OF E IS: /<15><12> /2. /<200>
52 000345 040 126 101
53 000350 114 125 105
54 000353 040 117 106
55 000356 040 105 040
56 000361 111 123 072
57 000364 015 012 062

```

SUM.MAC VERSION 1 MACRO V05.00 Saturday 08-Jan-83 08:07 Page 1-1

000367 056 200 .EVEN  
52 .END EXP  
53

SUM.MAC VERSION 1 MACRO V05.00 Saturday 08-Jan-83 08:07 Page 1-2

```

Symbol table
A 000124R FIFTH 000074R FOURTH 000052R N = 000106 THIRDD 000044R
EXP 000000R FIRST 000012R MESSAB 000342R SECONDD 000022R
ABS, 000000 000 (RW,I:GBL,ABS,DVR)
000372 001 (RW,I:LCL,REL,CON)
Errors detected: 0

```

\*\*\* Assembler statistics  
Work file reads: 0  
Work file writes: 0  
Size of work file: 8222 Words ( 33 Pages)  
Size of core pool: 15616 Words ( 61 Pages)  
Operating system: RT-11  
Elapsed time: 00:02:55.03  
DK:SUM,LP:SUM=DK:SUM/C

SUM.MAC VERSION 1 MACRO V05.00 Saturday 08-Jan-83 08:07 Page 5-1  
Cross reference table (CREF V05.00)

```

A 1-14 1-47#
EXP 1-11# 1-54
FIFTH 1-37# 1-39
FIRST 1-13# 1-44
FOURTH 1-25# 1-28
MESSAB 1-11 1-51#
N 1-7# 1-12
SECONDD 1-15# 1-21
THIRDD 1-23# 1-35

```

SUM\_HAC VERSION 1 MACRO V05.00 Saturday 08-Jan-83 08:07 Page H-1  
 Cross reference table (CREF V05.00)

.EXIT 1-3# 1-45  
 .PRINT 1-3# 1-11  
 .TTYOU 1-3# 1-41

.LINK/HAP SUM  
 RT-11 LINK V08.00 Load Map Saturday 08-Jan-83 08:11 Page 1  
 SUM .SAV Title: SUM.HA Ident:

Section	Addr	Size	Global Value	Global Value	Global Value
.ABS.	000000	001000	= 256.	words (RW,I:OBL,ABS,OVR)	
	001000	000372	= 125.	words (RW,I:LCL,REL,CDN)	

Transfer address = 001000, High limit = 001370 = 380. words

.RENAME MATCH.BAS MATCH.MAP

.MACRO/LIST/CROSSREFERENCE DEMDFG  
 DEMDFG MACRO V05.00 Saturday 08-Jan-83 08:11 Page 1

```

1          .TITLE DEMDFG
2          .IDENT /V05.00/
3          ; FOREGROUND DEMONSTRATION PROGRAM TO PRINT MESSAGE TO BACKGROUND, THEN
4          ; QUEUE A MESSAGE EVERY 2 SECONDS FOR THE BACKGROUND TO RING THE BELL.
5
6          .MCALL .SDATC,.PRINT,.HRKT,.GSET,.SPND
7
8          000000          START:11 .PRINT #MSG          ;PRINT INTRODUCTORY MESSAGE
9          000006          .GSET #QUEUE,#100.          ;SET ASIDE 100 0 ELEMENTS FOR MESSAGES
10         000020          .HRKT #AREA,#TIME,#HKTC,#1    ;SET UP MKTIM FOR 2 SECONDS FROM NOW
11         000054          .SPND                          ;SUSPEND THE FG TILL MKTIM SATISFIED
12
13         ;
14         ; MKTIM COMPLETION ROUTINE
15         000062 024727 000314 000132 HKTC:1 CMP MSGCNT,#90.          ;90 MESSAGES QUEUED YET?
16         000070 003020          BGT MKTC1            ;YES-NO SENSE QUEUEING ANOTHER
17         000072          .SDATC #AREA,#BUFFER,#1,#SDATC ;SEND MESSAGE TO BG
18         000126          INC MSGCNT                  ;BUMP MESSAGE COUNTER
19         000132          .HRKT #AREA,#TIME,#HKTC,#1    ;SET UP ANOTHER MKTIM FOR 2 SECONDS
20         000166 000207          RETURN                ;RETURN FROM COMPLETION ROUTINE
21
22         ;
23         ; SDATC COMPLETION ROUTINE
24         000170 005367 000206          SDATC:1 DEC MSGCNT          ;DONE OF THE MESSAGES HAS BEEN RECIEVED
25         000174 000207          RETURN                ;RETURN(RTS PC)
26
27         ;
28         ; ASCII MESSAGES
29         .NLST BEX
30         000176 106 117 122 MSG:1 .ASCII /FOREGROUND DEMONSTRATION PROGRAM/<15><12>
31         000240 123 105 116          .ASCII /SENDS A MESSAGE TO THE BACKGROUND PROGRAM 'DEMDFG'/'<15><12>
32         000324 105 126 105          .ASCII /EVERY 2 SECONDS, TELLING IT TO RING THE BELL./
33         .EVEN
34
35         000402 000000          MSGCNT:1 .WORD 0          ;MESSAGE COUNTER
36
37
38         000404 000000          TIME:1 .WORD 0          ;TIME CONSTANT
39         000406 000170          .WORD 40.*2          ;HIGH ORDER
40                                     ;60 TICKS A SECOND, 2 SECONDS
41
42         000410          AREA:1 .BLKW 4          ;ENT ARGUMENT AREA
43
44         000424          BUFFER:1 .BLKW 400          ;BUFFER FOR MESSAGES
45
46         001424          QUEUE:1 .BLKW 100.*10.      ;QUEUE AREA
47                                     ;10. WORDS PER QUEUE ELEMENT FOR THE XM MONITOR
48
49         000000'          .END START

```

DEMOFG MACRO V05.00 Saturday 08-Jan-83 08:11 Page 1-1

Symbol table

AREA	000410R	MKTC1	000132R	QUEUE	001424R	START	000000RG	...V1 = 000003
BUFFER	000424R	HSG	000174R	SBATC	000170R	TIME	000404R	...V2 = 000027
MKTC	000062R	HSGCNT	000402R					

.ABS. 000000 000 (RW,I:OBL,ABS,OVR)  
 005344 001 (RW,I:LCL,REL,CDN)  
 Errors detected: 0

\*\*\* Assembler statistics

Work file reads: 0  
 Work file writes: 0  
 Size of work file: 9386 Words ( 37 Pages)  
 Size of core pool: 15616 Words ( 61 Pages)  
 Operating system: RT-11

Elapsed time: 00:02:27.42  
 DK1:DEMDFG.LP:DEMDFG=DK1:DEMDFG/C

DEMOFG MACRO V05.00 Saturday 08-Jan-83 08:11 Page S-1  
 Cross reference table (CREF V05.00)

...	V1	1-9	1-10	1-17	1-19
...	V2	1-17	1-17	1-17#	1-17#
AREA	1-10	1-17	1-19	1-19	1-41#
BUFFER	1-17	1-43#			
MKTC	1-10	1-15#	1-19		
MKTC1	1-16	1-19#			
MSG	1-8	1-30#			
MSGCNT	1-15	1-19#	1-24#	1-35#	
QUEUE	1-9	1-46#			
SBATC	1-17	1-24#			
START	1-8#	1-48			
TIME	1-10	1-19	1-38#		

DEMOFG MACRO V05.00 Saturday 08-Jan-83 08:11 Page H-1  
 Cross reference table (CREF V05.00)

...	CM0	1-9								
...	CM1	1-17								
...	CM2	1-10	1-10	1-17	1-17	1-17	1-17	1-19	1-19	1-19
...	CM4	1-17								
...	CM5	1-9	1-10	1-17	1-19					
...	CM6	1-10	1-19							
.HRKT	1-6#	1-10	1-19							
.PRINT	1-6#	1-8								
.GSET	1-6#	1-9								
.SDATC	1-6#	1-17								
.SPND	1-6#	1-11								

.LINK/FOREGROUND/HAP DEMDFG  
 RT-11 LINK V08.00 Load Map Saturday 08-Jan-83 08:15 Page 1  
 DEMDFG.REL Title: DEMDFG Ident: V05.00

Section	Addr	Size	Global Value	Global Value	Global Value
.ABS.	000000	001000	= 256.	words (RW,I:OBL,ABS,OVR)	
	001000	005344	= 1394.	words (RW,I:LCL,REL,CDN)	
			START	001000	

Transfer address = 001000, High limit = 006342 = 1649. words

.DIRECTORY \*.OBJ  
 08-Jan-83  
 ODT .OBJ 8 22-Oct-82 VDT .OBJ 8 22-Oct-82  
 VTHDLR.OBJ 8 22-Oct-82 SYSLIB.OBJ 46 22-Oct-82  
 ERRDUT.OBJ 15 22-Oct-82 FORLIB.OBJ 161 12-May-80  
 TEST .OBJ 1 08-Jan-83 SUM .OBJ 1 08-Jan-83  
 DEMDFG.OBJ 1 08-Jan-83 GRAPH .OBJ 16 08-Jan-83  
 10 Files, 265 Blocks  
 354 Free blocks

.DELETE/NOQUERY GRAPH.\*

.DELETE/NOQUERY SUM.\*

.DELETE MATCH.MAP

.DEASSIGN  
.TIME  
08113144

**SUMMARY:  
COMMANDS TO  
START AN  
INDIRECT FILE**

**@filnam.COM**

Start the execution of the specified indirect file (filnam.COM).

**CTRL/C CTRL/C**

Halt execution of the indirect command file (use with caution).

**DEASSIGN**

Remove logical device assignments.

**FILE MAINTENANCE**

Indirect file INDCT.COM contains commands that perform the appropriate copy and delete file maintenance operations. If the commands were not already part of the file, you would need to perform the appropriate file maintenance commands, in monitor command mode, after execution.

**REFERENCE**

*RT-11 System User's Guide* (AA-5279C-TC). Maynard, Mass.: Digital Equipment Corporation, 1983.

A guide to the use of the RT-11 operating system.

## CHAPTER 17

### ADVICE TO NEW USERS

This manual introduces several common RT-11 functions but is neither exhaustive nor comprehensive in its treatment of system features, commands, or their options. For many users, this manual's description of these fundamental system operations is sufficient; other users, however, may need or want further description of a programming language, extended system features, or the internal workings of the RT-11 system. These people should consult the references at the end of each chapter, the *Guide to RT-11 Documentation*, *RT-11 System User's Guide*, or the *RT-11 System Utilities Manual*. The *Guide to RT-11 Documentation* lists all RT-11-related material available from DIGITAL; the *RT-11 System User's Guide* explains in detail monitor commands and command options; the *RT-11 System Utilities Manual* describes the use of the RT-11 system utilities to develop programs, execute programs, and maintain files and storage media.

The *Introduction to RT-11* has shown you the right way to use some important system features and their associated monitor commands. This information, combined with the following basic guidelines for using the system, can help you to avoid pitfalls common to new users:

- Do not become dependent on a single copy of a file. Always make a backup copy of any useful file.
- When using the editor, close files periodically to preserve edits. Divide long editing sessions into short ones so that user — or hardware — errors do not lose the efforts of long hours of editing. Close the file with the EX command and begin editing again from where you left off.
- Avoid careless use of wildcard operations that manipulate multiple files. Use the /QUERY option to verify the operation to be performed.
- When using indirect files or BATCH streams, avoid operations that manipulate any of the system (.SYS) files or the indirect file in use. Check the indirect file carefully for errors before you use it. Once the command stream is initiated, you may be unable to detect and prevent possibly serious errors.
- If you run two jobs under the control of the foreground/background monitor, be sure there is no conflict of nondirectory-structured devices (LP:, MT:, TT:) used by the two jobs.

## USING THE HELP FILE



A HELP file containing information about the keyboard monitor commands and how to use them is distributed with the RT-11 system. A list of keyboard monitor commands and a description of their functions can be displayed at the terminal by typing:

```
.HELP *(RET)
```

To get a detailed description of the use of the HELP command itself, type only HELP.

```
.HELP(RET)
```

The following information is displayed on your terminal.

```
HELP                Lists helpful information

SYNTAX
  HELP[/options][ topic[ subtopic[:items...]]...]
  or HELP *

SEMANTICS
  HELP * lists the items for which help is
  available.
  HELP lists the HELP text (of which this is a
  part).
  HELP topic lists information on the specific
  topic only.
  HELP topic subtopic lists information on the
  specific subtopic only (for example,
  HELP HELP SEMANTICS lists the paragraph of
  which this text is a part).
  HELP topic subtopic:item lists only the text
  associated with the specific item.
  HELP topic/item lists the text associated with
  the specific item under the subtopic OPTIONS.
  Valid topics are the Keyboard monitor commands.
  Subtopics are "SYNTAX", "SEMANTICS", "OPTIONS",
  and "EXAMPLES".
  Items are specific command options.

OPTIONS
  PRINTER
    Prints the HELP text on the line Printer
  TERMINAL (default)
    Types the HELP text on the terminal

EXAMPLES
  HELP COPY                !Lists information about
                           !COPY command
  HELP/PRINTER EXECUTE    !Prints information
                           !about EXECUTE command
  HELP PRINT OPTION:COPIES !Describes the COPIES
                           !option for PRINT
  HELP COPY/BOOT/DEVICE   !Describes the listed
                           !options for COPY
```

In the command syntax shown above, topic represents a specific keyboard monitor command about which you need information. The subtopic represents a specific category within a topic; the subtopics are syntax, semantics, options, and examples. The

item represents one of the members within the subtopic group. You can specify more than one item in the command line if you separate the items with a colon (:).

The only two options you can use with the HELP command are /PRINTER and /TERMINAL. The option /PRINTER sends the help information to a printer if one is available. The option /TERMINAL (the default mode) sends the information to the terminal.

To get all the information in the help file about the keyboard monitor command ASSIGN, type the following command:

```
.HELP ASSIGN@
```

You have used this command in examples in the other chapters. The following information is displayed at your terminal:

```
ASSIGN    Associates a logical device name with a
          physical device

SYNTAX
  ASSIGN physical-device-name logical-device-name

SEMANTICS
  Physical-device-name is the RT-11 standard
  permanent name for the device.
  Logical-device-name is one to three alphanumeric
  characters long with no intervening spaces or tabs.
  The physical name and logical name must be
  separated by a space.

OPTIONS
  None

EXAMPLES
  ASSIGN RK1: DK:
```

When you want specific information for a keyboard monitor command, such as the syntax, semantics, options, or examples, include that subtopic in the command. For example, the following command lists all the options that are available for use with the DIRECTORY keyboard monitor command:

```
.HELP DIRECTORY OPTIONS@
```

If you need information only about a specific item in a list of options, type the item in the command line.

```
.HELP DIRECTORY OPTIONS:ORDER@
```

```
ORDER[:category]
  Orders the directory listing according to the category
  specify; same as /SORT. Categories are:
  NAME- orders alphabetically by file name
  TYPE- orders alphabetically by file type
  SIZE- orders by file size
  DATE- orders by creation date
  POSITION- orders by file position on the device
```



## APPENDIX A MANUAL BOOTSTRAPPING OPERATIONS

PDP-11 computers that do not have the automatic bootstrapping capability described in Chapter 2 require manual bootstrapping. This appendix's first section, Bootstrapping the System, provides information that you will need to bootstrap the system manually. Follow the procedure described in this section; you will be referred — if appropriate for your system — to one of the other sections, which describe using a pushbutton console to bootstrap, using a switch register console to bootstrap, and typing the bootstrap.

Once you have identified your hardware configuration, you are ready to bootstrap the system. The purpose of the bootstrap procedure is to load and start the RT-11 monitor in computer memory, activating the RT-11 operating system for your use.

### BOOTSTRAPPING THE SYSTEM

1. Set the terminal to an on-line condition.
2. Make sure that the computer power is on and that the computer is not already in use. Stop the computer, following one of two procedures:
  - If your operator's console has switches, set the switches to HALT, then ENABLE.
  - If your operator's console has pushbuttons, locate the button labeled CNTRL; hold it down and push the button labeled HLT/SS; then release both.
3. Place the system volume in device unit 0. Make sure that the system volume is write-protected (for all except RX01 or RX02 diskettes, which are always write-enabled).
4. Place the storage volume in the device unit you identified for question 8 in the Hardware Configuration section of Chapter 2. Make sure that this volume is write-enabled.
5. If your operator's console has pushbuttons, continue to step 6. Otherwise, go to step 8.
6. Locate the pushbutton labeled CNTRL, hold it down and push the button labeled BOOT. Check the terminal printer or screen. If there is no response, read the Using a Pushbutton Console to Bootstrap section of this appendix; otherwise, continue to step 7.
7. Your terminal printer or screen should show several numbers and then a dollar sign (\$).

Type on the terminal keyboard the two-letter mnemonic that represents your system volume (from question 6 in the Hardware Configuration section<sup>1</sup>) followed by a carriage return. Be sure to use the SHIFT key so that you type uppercase characters. For example, for RX02 diskette, type:

```
#DY(RET)
```

Continue to step 11.

8. Check your switch console. If it has a three-way dial labeled DC OFF, DC ON, and STAND BY, go to step 9. If it has three individual switches labeled DC ON/OFF, ENABLE/HALT, and LTC ON/OFF, go to step 10. If it has a row of switches across the entire console, read the Using a Switch Register Console to Bootstrap section of this appendix.
9. Set the three-way dial to DC ON. Then locate the BOOT switch (to the left of the dial) and raise it. Go to step 11.
10. Put all three switches in the up position; then move the DC ON/OFF switch down and up and check the terminal response.

- If the terminal displays the dollar sign (\$), type on the keyboard the two-letter mnemonic that represents your system volume (from question 6 in the Hardware Configuration section) followed by a carriage return. Be sure to use the SHIFT key so that you type uppercase characters. For example, for RX02 diskette, type:

```
#DY(RET)
```

Continue to step 11.

- Any other response indicates that you must type the bootstrap on the terminal keyboard. Read the Typing the Bootstrap on the Terminal Keyboard section of this appendix.
11. If your system has been successfully bootstrapped, a message like the following prints on the console terminal.

```
RT-11FB  V05.xx (the xx's represent numbers that have  
significance only for DIGITAL's soft-  
ware development; you can ignore  
these numbers)
```

---

<sup>1</sup>The RK05 disk is an exception. Hardware bootstraps use DK, not RK, for RK05.

If the message indicating RT-11FB V05 does not appear, refer to Appendix B and read the section entitled Suggestions for Bootstrapping the System.

The message indicates that the foreground/background monitor component of the RT-11 Version 5.0 operating system is active. Set the system volume to a write-enabled condition (for all except RX01 and RX02 diskettes, which are always write-enabled).

You should now direct your attention to the console terminal, since system interaction continues on this device. Continue to Chapter 3.

The bootstrap for your RT-11 computer system consists of a series of six-digit numbers that you must type on the terminal keyboard. First, obtain the bootstrap from the *RT-11 Installation Guide*, and copy the numbers into the space below:

**TYPING THE  
BOOTSTRAP  
ON THE TERMINAL  
KEYBOARD**

Now, follow the instructions below to type, on your terminal keyboard, the bootstrap (if you make a mistake, type the DELETE key on the terminal keyboard, once for each typing error, and then type the correct digit[s]):

1. Type 001000.
2. Type slash (/).
3. Type the first number in the bootstrap column.
4. Type the LINE FEED key.
5. Type the next number in the bootstrap column.
6. Repeat steps 4 and 5 until you have typed all the numbers in the column.
7. Type the RETURN key.
8. Type 1000G.
9. Continue to step 11 in the section entitled Bootstrapping the System in this appendix.

## USING A PUSHBUTTON CONSOLE TO BOOTSTRAP

If your computer has a pushbutton console similar to the one shown in Figure A-1, you can use the buttons to manually give the computer the information it needs to bootstrap the system.

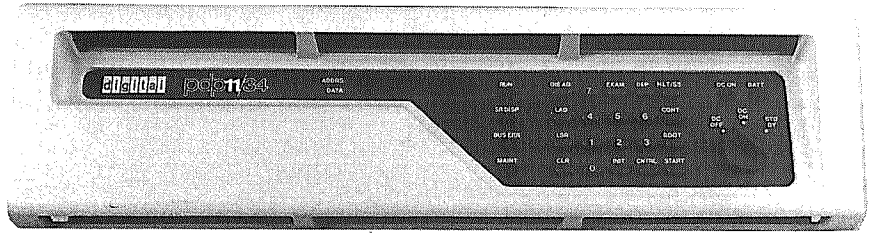


Figure A-1 Pushbutton Console

The bootstrap for your RT-11 computer system consists of a series of six-digit numbers that you must load into the computer using the pushbutton console. First, obtain the bootstrap of your system device from the *RT-11 Installation Guide*, and copy the numbers into the space provided below. If your system has a hardware bootstrap,<sup>1</sup> the bootstrap will consist of only two numbers, which you should copy into the left-hand space; otherwise, the bootstrap will consist of two columns of numbers labeled Location and Contents, which you should copy into the right-hand space:

Hardware Bootstrap

Other Bootstraps

Load Address =

Start Address =

To activate the hardware bootstrap, use the pushbuttons to set the numbers as described in the following steps (if you make a mistake, push the button labeled CLR, then reenter the number):

1. Push the appropriate buttons for the load address (read the number from left to right).
2. Push LAD.
3. Push the appropriate buttons for the start address (read the number from left to right).
4. Push the button labeled CNTRL, and, while holding it down, push the button labeled START.
5. Continue to step 11 in the Bootstrapping the System section of this appendix.

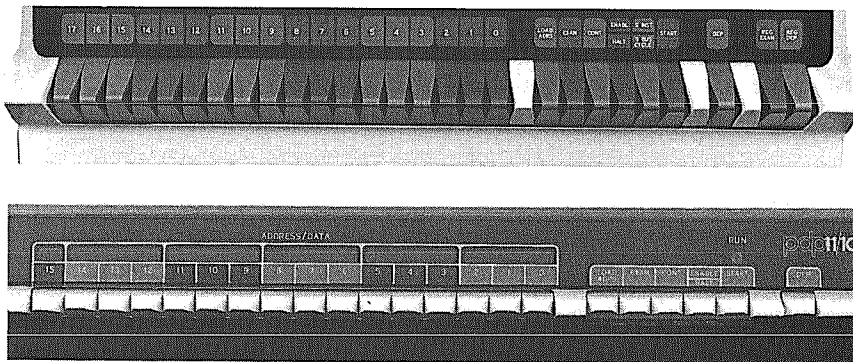
<sup>1</sup>A hardware bootstrap is bootstrapping information that is already in computer memory but that you must activate by entering a load address and a start address, each a six-digit number.

To activate other bootstraps, use the pushbuttons to set the numbers as described in the following steps (if you make a mistake, push the button labeled CLR, then reenter the number):

1. Push 1000 (read the number from left to right).
2. Push LAD.
3. Push the appropriate buttons for the first number in the Contents column (read the number from left to right).
4. Push DEP; push CLR.
5. Push the appropriate buttons for the next number in the Contents column (read the number from left to right).
6. Repeat steps 4 and 5 until all numbers in the column have been used.
7. Push 1000.
8. Push LAD.
9. Push the button labeled CNTRL, and, while holding it down, push the button labeled START.
10. Continue to step 11 in the Bootstrapping the System section of this appendix.

If your computer has a switch register console similar to the one shown in Figure A-2, you can use the switches to manually give the computer the bootstrapping information it needs to start the system.

### USING A SWITCH REGISTER CONSOLE TO BOOTSTRAP



**Figure A-2 Switch Register Consoles**

Several switches on the console are spring-loaded. This means that the switch moves in only one direction and returns to its initial position after you use it. You must set the remaining switches either up or down as instructed.



NOTE

The switch register is the group of switches appearing on the left of the console. Your switch register may have only 16 switches rather than 18; in this case you can ignore the left-hand two digits of the binary number when you set the switches.

To activate the hardware bootstrap:

1. Set the switch register to the appropriate positions for the load address.
2. Press the spring-loaded LOAD ADDR switch.
3. Set the switch register to the appropriate positions for the start address.
4. Press the spring-loaded START switch.
5. Continue to step 11 in the Bootstrapping the System section of this appendix.

To activate other bootstraps, set the numbers into the switch register using the following method:

1. Set the switch register to the appropriate positions for the number 001000.
2. Press the spring-loaded LOAD ADDR switch.
3. Set the switch register to the appropriate positions for the first number in the Contents column.
4. Press the spring-loaded DEP switch.
5. Set the switch register to the appropriate positions for the next number in the Contents column.
6. Repeat steps 4 and 5 until all the numbers in the column have been used.
7. Set the switch register to the appropriate positions for the number 001000.
8. Press the spring-loaded LOAD ADDR switch.
9. Press the spring-loaded START switch.
10. Continue to step 11 in the Bootstrapping the System section of this appendix.



## APPENDIX B SELECTED SYSTEM TOPICS

The remarks in this appendix cover a variety of topics that should prove helpful to you as you perform the demonstrations in the manual. Included, for example, are instructions for starting and stopping the system, alternate methods for performing some system operations, and directions for using the language volume. The sections are listed here in the order in which they are referenced from within the text of the manual.

You can plan to take a break at the end of any individual chapter in this manual. If you intend to be away from the computer system for any length of time, you should halt the system and remove your system and storage volumes.

### STOPPING AND STARTING THE SYSTEM

Perform the following steps in order:

#### Stopping the System

1. Stop the computer.
  - If your computer is a PDP-11/23-PLUS, lift the HALT toggle switch to the up position.
  - If your computer is a PDP-11/24 or a PDP-11/44, push the HALT/CONT/BOOT horizontal toggle switch to the HALT position.
  - If your computer operator's console has switches, press the HALT switch.
  - If your computer operator's console has pushbuttons, hold the CNTRL button down and push the HLT/SS button.
2. Unload the system volume. Set the device unit to an off-line condition, and remove the system volume.
3. Unload the storage volume. Set the device unit to an off-line condition, and remove the storage volume.
4. Remove and save all terminal and line printer output listings.

Perform the following steps in order:

#### Starting the System

1. Follow the bootstrap procedure, as described in Chapter 2 or Appendix A.

2. Enter the current date and time-of-day (Chapter 4).
3. Make any necessary logical device assignments. For the examples in this manual, you must assign the logical name VOL: to your storage volume (Chapter 4).

## **THE SYSTEM STOPS UNEXPECTEDLY**

If for any reason the computer system stops unexpectedly, request help from an experienced user if possible. The problem may be accompanied by an error message (see the *RT-11 System Message Manual*); the problem may be in the hardware, in the software, or in your program. Once the problem is diagnosed and corrected, try to bootstrap the system again.

## **SUGGESTIONS FOR BOOTSTRAPPING THE SYSTEM**

You must be able to bootstrap your RT-11 system before you can perform the demonstrations in this manual. Three common bootstrapping problems and suggestions for their correction follow.

1. You cannot remember how to bootstrap your system.

Ask an experienced RT-11 user to help you. If no one is available, read the bootstrapping instructions in the *RT-11 Automatic Installation Booklet* or the *RT-11 Installation Guide*. If necessary consult the appropriate hardware manuals for the devices that are part of your system; these manuals provide a description of the device and operating procedures. Then try the bootstrap procedures again.

2. You have followed the bootstrapping instructions correctly, but your system printed a message other than what you expected.

- a. The message can be one of the following:

?BOOT-F-No boot on volume

?BOOT-U-Conflicting SYSGEN options

?BOOT-U-Handler file not found

?BOOT-U-Insufficient memory

?BOOT-U-I/O error

?BOOT-U-Monitor file not found

?BOOT-U-No KT11

?BOOT-U-Swap file too small

?BOOT-U-SWAP.SYS not found

```
?BOOT-W-Error reading handler
```

```
?BOOT-W-Invalid or missing TT.SYS
```

These are bootstrap error messages, indicating that a problem in the system is preventing bootstrapping. These eleven messages are fully explained in the *RT-11 System Message Manual*, but you should not try to correct the problem yourself if an experienced user is available to help.

- b. The message can be one of the following:

```
RT-11SJ      V05-xx
```

```
RT-11XM      V05-xx
```

These indicate that a valid RT-11 V5 monitor program has been bootstrapped, but it is not the one you should be using. Reboot the correct monitor program by typing the following commands on the terminal:

```
.BOOT(RET)  
Device or file? RT11FB.SYS(RET)
```

- c. Any other message indicates that an old version of RT-11 (V1, V2, V2B, V2C, V3, V3B, or V4) has been bootstrapped. Only Version 5 of RT-11 can be used to perform the demonstrations in this manual.
3. You followed the bootstrapping instructions correctly, but nothing happened, that is, there was no terminal response at all.

Repeat the bootstrap procedure from the beginning. Before you begin, make sure that the system volume is properly mounted in device unit 0. The computer should be on but not running (the light labeled RUN should not be on); if the computer is running, stop it as described above. Make sure that the terminal is on line and that its baud rate is set to 300. If you are using a display terminal, make sure that the screen is bright enough for you to read. If your terminal uses a paper printer, make sure that the paper is properly loaded.

A copy of the RT-11 Version 5 system volume, as distributed by DIGITAL, should be stored away for safekeeping as a backup copy. If you do not have a backup copy of your system volume, create one before you continue. Manual backup instructions are in the *RT-11 Installation Guide*; an experienced user should perform the backup operation.

## BACKING UP THE SYSTEM VOLUME

## **DIRECTORY- VS NONDIRECTORY- STRUCTURED VOLUMES**

Storage volumes are called file-structured volumes because they are capable of physically storing files. These volumes can be further categorized as directory-structured and nondirectory-structured; the distinction is based on the method of directory information storage, collection, and printing.

Directory information includes file names and types, dates of creation, and (in most cases) file lengths. When you type the **DIRECTORY** command, this directory information prints on your terminal. Volumes such as disks and diskettes keep this information in a single place at the beginning of the volume. Each time you add or delete a file, the directory information is updated. These volumes, which maintain a directory separately from the files described, are said to be directory-structured. Magtape volumes, on the other hand, do not keep directory information in any single, separate place on the tape but rather with each individual file. For these volumes the directory information is collected for printing as each file is encountered during a sequential reading of all files on the tape. Thus, these volumes are said to be nondirectory-structured.

You can list the directory from either type of volume in complete or abbreviated format. Complete directories include the file name, file type, file length, and date of creation (if the **DATE** command was used before the file's creation). For most volumes, the directory format is as follows:

```
B-Jan-83  
FILE  .TYP    26  21-Feb-80
```

Abbreviated directories include only the file name and file type, and are printed in five columns. For more information about directory-structured and nondirectory-structured volumes, see the *RT-11 System User's Guide*.

## **ALTERNATE RENAME OPERATION FOR MAGTAPE USERS**

You cannot use the **RENAME** monitor command if your volume is a magtape because of the magtape's sequential (nondirectory-structured) nature. To perform the **RENAME** operation, you must first copy the file, using the new file name, and then delete the old file.

For example, to change the name of the **GRAPH.TWO** file, located on your storage volume, to **GRAPH.FOR**, first make a copy of **GRAPH.TWO**, giving the new file the name **GRAPH.FOR**.

### **Long Command Format**

```
.COPY(RET)  
From? VOL:GRAPH.TWO(RET)  
To ? GRAPH.FOR(RET)
```

**Short Command Format**

```
.COPY VOL:GRAPH.TWO GRAPH.FOR(RET)
```

You now have two copies of the GRAPH file. Delete the one not wanted, using the monitor DELETE command. (This command is described in Chapter 7 in the section entitled File Delete Operations.)

**Long Command Format**

```
.DELETE(RET)
Files? VOL:GRAPH.TWO(RET)
```

**Short Command Format**

```
.DELETE VOL:GRAPH.TWO(RET)
```

A single copy of GRAPH.FOR now resides on your system volume. Copy the file onto your storage volume.

**Long Command Format**

```
.COPY(RET)
From? GRAPH.FOR(RET)
To ? VOL:GRAPH.FOR(RET)
```

**Short Command Format**

```
.COPY GRAPH.FOR VOL:GRAPH.FOR(RET)
```

Delete the original file.

**Long Command Format**

```
.DELETE(RET)
Files? GRAPH.FOR(RET)
```

**Short Command Format**

```
.DELETE GRAPH.FOR(RET)
```

The combined effect of these four commands is to “rename” GRAPH.TWO to GRAPH.FOR.

The FORTRAN/BASIC language volume was created during system installation specifically for your use with this manual. This volume contains the FORTRAN IV and/or BASIC-11 language processors and the monitor files required to use these language processors. Before you can perform the FORTRAN IV or BASIC-11 demonstrations, you must substitute this

**USING THE  
FORTRAN/BASIC  
LANGUAGE VOLUME**

FORTRAN/BASIC language volume for the system volume currently mounted in device unit 0. The language volume then serves as the system volume during the course of the FORTRAN IV and BASIC-11 demonstrations.

Make sure that no system operations are in progress (the monitor prompt, the period, should appear at the left margin of the terminal printer), and stop the system (see *Stopping and Starting the System*, this appendix). Now remove the system volume currently loaded in device unit 0, and insert and write-protect the language volume. Bootstrap the system (see *Stopping and Starting the System*, this appendix). The following monitor message should appear:

```
RT-11FB      V05.xx
```

Write-enable the volume. Then enter the current date and time-of-day, and assign the logical name VOL: to your storage volume, just as you did in Chapter 4. When you have done this, you are ready to run the language demonstration. Return to the main text of this manual.

## **SUBSTITUTING VOLUMES DURING OPERATIONS**

Users of FORTRAN IV on diskette who have the FORTRAN IV language processor on a volume apart from their system volume must occasionally copy files and substitute volumes. These operations are necessary when files needed are not stored on a currently mounted volume. The appropriate volume, containing needed files, must be substituted for a currently mounted volume. If the volume to be dismounted contains necessary files, these files must be copied to a volume that will remain mounted.

For example, before you can compile the FORTRAN IV file THIRD.FOR, you must substitute the language volume containing the FORTRAN IV compiler for the system volume currently loaded in device unit 0. First, however, you must copy the file THIRD.FOR to your storage volume so that it will be available for use.

### **Long Command Format**

```
.COPY(RET)  
From? THIRD.FOR(RET)  
To ? VOL:THIRD.FOR(RET)
```

### **Short Command Format**

```
.COPY THIRD.FOR VOL:THIRD.FOR(RET)
```

When the copy operation is finished, stop the system, remove the system volume currently loaded in unit 0, and insert and write-protect the language volume. See Stopping and Starting the System (this appendix) if necessary. The following message appears when the language volume is bootstrapped.

```
RT-11FB    V05,xx
```

Write-enable the volume. Then enter the current date and time-of-day, and assign logical name VOL: to your storage volume, as described in Chapter 4.

Next, compile the FORTRAN IV program THIRD.FOR, which is now on VOL:.

### Long Command Format

```
.FORTRAN␣
Files? VOL:THIRD.FOR␣
PUTSTR
```

### Short Command Format

```
.FORTRAN VOL:THIRD␣
PUTSTR
```

The FORTRAN command causes the object module to be created on the default storage volume, which is presently the system volume (that is, the language volume). Any errors that occur during the compile operation indicate that the source file, THIRD.FOR, contains typographical errors. You must edit the file to correct any errors, recompile, and then copy the file to VOL:. Once you have an object module compiled without error and stored on VOL:, reload the main system volume in unit 0. Follow the directions in Stopping and Starting the System. Bootstrap and write-enable the system volume, enter the current date and time-of-day, and assign the logical name VOL: to your storage volume.

Now copy the object module on VOL: back to the system volume.

### Long Command Format

```
.COPY␣
From? VOL:THIRD OBJ␣
To ? THIRD.OBJ␣
```

### Short Command Format

```
.COPY VOL:THIRD.OBJ THIRD.OBJ␣
```

Continue to Chapter 13, to the section entitled Building the Object Library.

## **USING THE LINK VOLUME**

The LINK volume was created during system installation for you to use with this manual. This volume contains the linker, LINK.SAV, and the system subroutine library, SYSLIB.OBJ. Before you can perform the linking demonstrations in Chapters 9 and 12, you must substitute this LINK volume for your current system volume, which is mounted in device unit 0. The LINK volume then serves as the system volume during the course of the linking demonstration.

First, transfer the object file you need to link to the storage volume.

### **Long Command Format**

```
.COPY␣  
From? GRAPH.OBJ␣  
To ? VOL:GRAPH.OBJ␣
```

### **Short Command Format**

```
.COPY GRAPH.OBJ VOL:GRAPH.OBJ␣
```

Make sure that no system operations are in progress (the monitor prompt, the period, should appear at the left margin of the terminal printer), and stop the system (see *Stopping and Starting the System*, this appendix). Now remove the system volume currently loaded in device unit 0, and insert and write-protect the LINK volume. Bootstrap the system. The following monitor message should appear:

```
RT-11FB    V05.XX
```

Write-enable the volume. Then enter the current date and time, and assign the logical name VOL: to your storage volume, just as you did in Chapter 4.

Finally, transfer the object file from the storage volume to the system volume.

### **Long Command Format**

```
.COPY␣  
From? VOL:GRAPH.OBJ␣  
To ? GRAPH.OBJ␣
```

### **Short Command Format**

```
.COPY VOL:GRAPH.OBJ GRAPH.OBJ␣
```

When you have done this, you are ready to run the linking demonstration. Return to the main text of this manual.

## FORTRAN/LINK FILE MAINTENANCE

Follow the file maintenance operations outlined in this section if you substituted both a FORTRAN IV language volume and a LINK volume to perform the demonstrations in Chapter 9.

First, mount the FORTRAN IV language volume in device unit 0. If you do not remember how to do this, follow the instructions in the section of this appendix entitled Using the FORTRAN/BASIC Language Volume.

Next, obtain a directory listing of all the files on your FORTRAN IV volume that have the name GRAPH, regardless of file type; these files were generated as a result of the exercises in Chapter 9.

### Long and Short Command Formats

```

•DIRECTORY GRAPH,*(RET)
  8-Jan-83
GRAPH .BAK      2  08-Jan-83  GRAPH .FOR      2  08-Jan-83
GRAPH .OBJ     16  08-Jan-83  GRAPH .LST      8  08-Jan-83
  4 Files, 28 Blocks
  48 Free blocks

```

Since you have corrected errors in the source file GRAPH.FOR, the version on your storage volume is obsolete. Transfer the corrected GRAPH.FOR file from your system volume to VOL:, thus replacing the obsolete file.

### Long Command Format

```

•COPY(RET)
From? GRAPH.FOR(RET)
To ? VOL:GRAPH.FOR(RET)

```

### Short Command Format

```

•COPY GRAPH.FOR VOL:GRAPH.FOR(RET)

```

Next, transfer GRAPH.LST to your storage volume. This enables you to examine the listing without having to recompile the program.

### Long Command Format

```

•COPY(RET)
From? GRAPH.LST(RET)
To ? VOL:GRAPH.LST(RET)

```

### Short Command Format

```

•COPY GRAPH.LST VOL:GRAPH.LST(RET)

```

Once you have transferred all valuable files to your storage volume, delete the unnecessary files from the system volume.

## Long Command Format

```
,DELETE(RET)  
Files? GRAPH.*(RET)  
Files deleted:  
DK.GRAPH.BAK ? Y(RET)  
DK.GRAPH.FOR ? Y(RET)  
DK.GRAPH.OBJ ? Y(RET)  
DK.GRAPH.LST ? Y(RET)
```

## Short Command Format

```
,DELETE GRAPH.*(RET)  
Files deleted:  
DK.GRAPH.BAK ? Y(RET)  
DK.GRAPH.FOR ? Y(RET)  
DK.GRAPH.OBJ ? Y(RET)  
DK.GRAPH.LST ? Y(RET)
```

Make sure that no system operations are in progress (the monitor prompt, the period, should appear at the left margin of the terminal printer), and stop the system (see *Stopping and Starting the System*, this appendix). Now remove the system volume currently loaded in device unit 0, and insert and write-protect the LINK volume. Bootstrap the system (see *Stopping and Starting the System*, this appendix). The following monitor message should appear:

```
RT-11FB      V05,xx
```

Write-enable the volume. Then enter the current date and time, and assign the logical name VOL: to your storage volume, just as you did in Chapter 4.

Obtain a directory of all files on the system volume that have the name GRAPH, regardless of file type; these files were created as a result of the linking demonstrations in Chapter 9.

## Long and Short Command Formats

```
,DIRECTORY GRAPH.*(RET)  
8-Jan-83  
GRAPH ,OBJ 14 08-Jan-83 GRAPH ,SAV 19 08-Jan-83  
2 Files, 33 Blocks  
80 Free blocks
```

Transfer GRAPH.SAV to your storage volume. Because GRAPH.SAV is an executable file, you can run the program without relinking it.

## Long Command Format

```
,COPY(RET)  
From? GRAPH.SAV(RET)  
To ? VOL:GRAPH.SAV(RET)
```

**Short Command Format**

```
.COPY GRAPH.SAV VOL:GRAPH.SAV(RET)
```

Next, delete the unnecessary files from your system volume.

**Long Command Format**

```
.DELETE(RET)  
Files? GRAPH.OBJ,GRAPH.SAV(RET)
```

**Short Command Format**

```
.DELETE GRAPH.OBJ,GRAPH.SAV(RET)
```

Finally, list the up-to-date directory of your storage volume so that you can see its current status.

```
.DIRECTORY VOL:(RET)
```

Leave the LINK volume mounted in device unit 0, and proceed to Chapter 12, Linking Object Programs.



# Glossary

**Absolute address**

The binary number that is assigned as the address of a physical memory storage location.

**Absolute section**

The portion of a program in which the programmer has specified physical memory locations of data items.

**Access time**

The interval between the instant at which data is requested from or for a storage device and the instant at which the data actually begins moving to or from the device.

**ADC (Analog to Digital Converter)**

A circuit that converts analog (voltage) signals to binary data.

**Address**

A label, name, or number that designates a location in memory where information is stored.

**Algorithm**

A prescribed set of well-defined rules or processes for the solution of a problem in a finite number of steps.

**Alphanumeric**

The subset of ASCII characters including the 26 alphabetic characters and the 10 numeric characters.

**ANSI**

American National Standards Institute.

**Application program (or package)**

A program that performs a function specific to the needs of a particular end-user or class of end-users. An application program can be any program that is not part of the basic operating system.

**Argument**

A variable or constant value supplied with a command that controls the command's action, specifically its location, direction, or range.

**Array**

An ordered arrangement of subscripted variables.

**ASCII**

The American Standard Code for Information Interchange; a standard code consisting of eight-bit coded characters for upper- and lower-case letters, numbers, punctuation, and special communication control characters.

**Assembler**

A program that translates symbolic source code into machine instructions. This program replaces symbolic operation codes with binary operation codes and symbolic addresses with absolute or relocatable addresses.

**Assembly language**

A symbolic programming language that can be translated directly into machine language instructions and is specific to a given type of control processing unit.

**Assembly listing**

A listing, produced by an assembler, that shows the symbolic code written by a programmer next to a representation of the actual machine instructions generated.

**Asynchronous**

The type of operation that is triggered by another event, as opposed to synchronous, or occurring at set time intervals.

**Background program**

A program that runs at a low priority, that is, when a higher priority (foreground) program is not using system resources.

**Backup file**

A copy of a file, created as a precaution against loss of the primary file.

**Base address**

An address used as the basis for computing the value of some other relative address; the address of the first location of a program or data area.

**BASIC-11 (Beginner's All-purpose Symbolic Instruction Code)**

An interactive, algebraic computer language that combines English words and decimal numbers. It is a widely available, standardized, simple beginner's language capable of handling industry and business applications.

**Batch processing**

A processing method in which programs are run consecutively without operator intervention.

**Baud**

A unit of measurement of transmission speed; bits per second.

**Binary**

The number system with a base of two; used by the internal logic of all digital computers.

**Binary code**

A code that uses two distinct characters, usually the numbers 0 and 1.

**Bit**

A binary digit. The smallest unit of information in a binary system of notation. It corresponds to a 1 or 0 and to one digit position in a physical memory word.

**Block**

A group of physically adjacent words or bytes of a size that is specific to a device. For input/output operations, the smallest addressable unit on a mass storage device.

**Bootstrap**

A technique or routine whose first instructions are sufficient to start a system of programs that bring an operating system into memory.

**BOT (Beginning Of Tape)**

A reflective marker that is applied to the backside of magtape and identifies the beginning of the magtape's recordable surface.

**Bottom address**

The lowest memory address into which a program is loaded.

**Breakpoint**

A location at which program operation is suspended to allow operator investigation.

**Buffer**

A storage area used to temporarily hold information being transferred between two devices or between a device and memory. A buffer is often a special register or a designated area of memory.

**Bug**

A flaw in the design or implementation of a program; a problem that can cause erroneous results.

**Bus**

A flat, flexible cable consisting of many transmission lines, or wires. It interconnects computer system components to provide communication paths for addresses, data, and control information.

**Byte**

The smallest memory-addressable unit of information. In a PDP-11 computer system, a byte is equivalent to eight bits.

**Call**

A transfer from one part of a program to another with the ability to return to the original program at the point of the call.

**Calling sequence**

A specified arrangement of the instructions and data necessary to pass parameters and control to a given subroutine.

**Central processing unit (CPU)**

A hardware unit of a computer that includes main memory and the registers and circuits that control the interpretation and execution of instructions.

**Character**

A single letter, numeral, or symbol used to represent information.

**Character pointer**

The place where the next character typed will be entered. During editing, the character pointer indicates the place in an ASCII text file where the next character typed will be entered into the file.

**Clear**

To delete the contents of a storage location by replacing the contents, usually with 0s or spaces.

**Clock**

A device within a computer system that keeps time, counts pulses, measures frequency, or generates regular periodic signals for synchronization.

**Code**

A system of symbols used to represent data or instructions that are executed by a computer.

**Coding**

The writing of instructions for a computer, using a system of symbols that is meaningful to a computer, an assembler, a compiler, or a language processor.

**Command**

A word, mnemonic, or character that, by virtue of its syntax in an input line, causes a computer system to perform a predefined operation.

**Command language**

The vocabulary used by a program or set of programs that directs the computer system to perform predefined operations.

**Command language interpreter**

The program that translates a predefined set of commands into instructions that a computer system can interpret.

**Command string**

A line of input entered into a computer system that generally includes a command, one or more file specifications, and optional qualifiers.

**Compile**

To produce binary code from the symbolic instructions of a high-level source language.

**Compiler**

A program that translates a high-level source language into machine instructions.

**Computer**

A machine that can be programmed to execute a set of instructions.

**Computer program**

A plan or routine for solving a problem on a computer.

**Computer system**

A data processing system that consists of hardware devices, software programs, and documentation that describes the operation of the system.

**Concatenation**

The joining of two or more strings of characters to produce a single string.

**Conditional assembly**

The assembly of certain parts of a symbolic program that occurs only when certain conditions are met during the assembly process.

**Configuration**

A selection of hardware devices, software routines, or programs that function together.

**Console terminal**

A keyboard terminal that acts as the primary interface between the computer operator and the computer system. The console terminal is used to initiate and direct system operations by running software on the computer.

**Constant**

A value that remains the same throughout a distinct operation. (Compare with Variable.)

**Context switching**

The saving of key registers and other memory areas before switching between jobs with different modes of execution. An example of context switching is the use of foreground/background programming.

**Conversational**

See Interactive.

**CPU**

See Central processing unit.

**Crash**

A hardware crash is the failure of a particular device to operate; the operation of an entire computer system may be affected. A software crash is the result of an operating system malfunctioning; the system's protection mechanisms may have failed or the software may not have executed correctly.

**Create**

To open, write data to, and close a file for the first time.

**Cross-reference listing**

A printed listing that identifies all references in a program to each specific symbol in a program. It includes a list of all the symbols used in a source program and the statements where the symbols are defined or used.

**Current location counter**

A counter kept by an assembler to determine the address assigned to an instruction or constant being assembled.

**Data**

A term used to denote facts, numbers, letters, and symbols. Data are the basic elements of information that can be processed by a computer.

**Data base**

An organized collection of interrelated data items that allow one or more applications to process the items, while disregarding physical storage locations.

**Data collection**

To bring data from one or more locations to a central location for eventual processing.

**Debug**

To detect, locate, and correct coding or logic errors in a computer program.

**Default**

The value of an argument, operand, or field assumed by a program if not specifically supplied by the user.

**Define**

To assign a value to a variable or constant.

**Delimiter**

A character that separates, terminates, or organizes elements of a character string, statement, or program.

**Device**

A hardware unit such as an I/O peripheral, magnetic tape drive, or line printer.

**Device control unit**

A hardware unit that electronically supervises one or more of the same type of devices. It acts as the link between the computer and the I/O devices.

**Device handler**

A routine that services and controls the hardware activities of an I/O device.

**Device independence**

The ability to program I/O operations independently of the device for which the I/O is intended.

**Device name**

A unique name that identifies each device unit on a system. It consists of a two-letter device mnemonic followed by an optional device unit number and a colon. For example, the common device name for RL02 disk drive unit 1 is DL1:.

**Device unit**

One of a set of similar peripheral devices. An example of a device unit is disk unit 0. It may be used synonymously with volume.

**Diagnostics**

A set of procedures used to detect and isolate malfunctions and mistakes.

**Digit**

A character used to represent one of the non-negative integers smaller than the radix (for example, in decimal notation, one of the characters 0 to 9; in octal notation, one of the characters 0 to 7; in binary notation, one of the characters 0 and 1).

**Direct access**

See Random access.

**Directive**

Assembler directives are mnemonics in an assembly language source program that are recognized by the assembler as commands to control a specific assembly process.

**Directory**

A file in the form of a table containing the names of and pointers to files on a mass storage volume.

**Directory-structured**

A storage volume is directory structured if the directory at the beginning of the volume contains information (file name, file type, length, and date-of-creation) about all the files on the volume. Such volumes include all disks, diskettes, and DECtapes.

**Disk device**

An auxiliary storage device on which information can be read or written.

**Display**

A peripheral device used to represent data graphically; normally refers to some type of cathode-ray tube system.

**Downtime**

The time interval during which a device or system is inoperative.

**Echo**

The printing of characters typed by the programmer on an I/O device such as a terminal.

**Edit**

To arrange and/or modify the format of data; for example, to insert or delete characters.

**Editor**

A program that allows the user to enter text into the computer and edit it. Editors are language-independent and will edit anything in character representation.

**Effective address**

The address used in the execution of a computer instruction.

**Emulator**

A hardware device that permits a program written for a specific computer system to be run on a different type of computer system.

**Entry point**

A location in a subroutine to which program control is transferred when the subroutine is called.

**EOT (End Of Tape)**

A reflective marker applied to the backside of magtape, which precedes the end of the reel.

**Error**

Any discrepancy between a computed, observed, or measured quantity and the specified value or condition.

**Execute**

To perform an instruction or run a program on the computer.

**Expression**

A combination of operands and operators that can be evaluated to a distinct result by a computing system.

**Extension**

The synonym used for file type.

**External storage**

A storage medium other than main memory, for example, a disk or tape.

**Field**

A specified area of a record used for a particular category of data.

**FIFO (First In/First Out)**

A data manipulation method in which the first item stored is the first item processed.

**File**

A logical collection of data that is treated as a unit, occupies one or more blocks on a mass storage volume, and has an associated file name and type.

**File maintenance**

The activity of keeping a mass storage volume and its directory up to date by adding, changing, or deleting files.

**File name**

The alphanumeric character string assigned by a user to identify a file. It can be read by both an operating system and a user. A file name has a fixed maximum length that is system-dependent. (The maximum length in an RT-11 operating system is six characters, the first of which must be alphabetic. Spaces are not allowed.)

**File specification**

A name that uniquely identifies a file maintained in any operating system. A file specification generally consists of at least three components: a device name, a file name, and a file type.

**File-structured device**

A device on which data is organized into files. The device usually contains a directory of the files stored on the volume. (For example, a disk is a file-structured device, but a line printer is not.)

**File type**

The alphanumeric character string assigned to a file either by an operating system or a user. It can be read by both the operating system and the user. System-recognizable file types are used to identify files having the same format or type. If present in a file specification, a file type follows the file name in a file specification, separated from the file name by a period. A file type has a

fixed maximum length that is system-dependent. The maximum in an RT-11 operating system is three characters, not including any spaces and excluding the preceding period.

**Flag**

A variable or register used to record the status of a program or device; the detection of errors by a translating program.

**Floating point**

A number system in which the position of the radix point is indicated by the exponent part of a number and another part represents the significant digits or fractional portion of a number (for example,  $5.39 \times 10^8$  — Decimal;  $137.3 \times 8^4$  — Octal;  $101.10 \times 2^{13}$  — Binary).

**Flowchart**

A graphical representation for the definition, analysis, or solution of a problem, in which symbols are used to represent operations, data, flow, and equipment.

**Foreground**

The area in memory designated for use by a high-priority program. The program that gains the use of machine facilities immediately upon request.

**FORTRAN IV (FORmula TRANslation)**

A problem-oriented language designed to permit scientists and engineers to express mathematical operations in a form with which they are familiar. It is also used in a variety of applications, including process control, information retrieval, and commercial data processing.

**Full duplex**

In communication, pertaining to a simultaneous, two-way, independent, asynchronous transmission.

**Function**

An algorithm, accessible by name and contained in the system software, that performs commonly used operations. For example, the square root calculation function.

**General register**

One of eight 16-bit internal registers in the PDP-11 computer. These are used for temporary storage of data.

**Global**

A value defined in one program module and used in others. Globals are often referred to as entry points in

**Interrupt vector**

Two words containing the address of an interrupt service routine and the processor state at which that routine is to execute.

**Iteration**

Repetition of a group of instructions.

**Job**

A group of data and control statements that does a unit of work. A program and all of its related subroutines, data, and control statements is an example; also, a batch control file.

**Label**

One or more characters used to identify a source language statement or line.

**Latency**

The time from the initiation of a transfer operation to the beginning of actual transfer; that is, verification plus search time. The delay while waiting for a rotating memory to reach a given location.

**Library**

A file containing one or more macro definitions or one or more relocatable object modules that are routines that can be incorporated into other programs.

**LIFO (Last In/First Out)**

A data manipulation method in which the last item stored is the first item processed; a push-down stack.

**Light pen**

A device, resembling a pencil or stylus, that can detect a fluorescent cathode-ray tube (CRT) screen. The pen is used to input information to a CRT display system.

**Linkage**

The code that connects two separately coded routines and passes values and/or control between them.

**Linked file**

A file whose blocks are joined together by references rather than by consecutive locations.

**Linker**

A program that combines many relocatable object modules into an executable module. It satisfies global references and combines program sections.

**Listing**

The printed copy generated by a line printer or terminal.

**Load**

To store a program or data in memory. To place a volume on a device unit and put the unit on line.

**Load map**

A table, produced by a linker, that provides information about a load module's characteristics; for example, the transfer address, the global symbol values, and the low and high limits of the relocatable code.

**Load module**

A program in a format that is ready for loading and executing.

**Location**

An address in storage or memory where a unit of data or an instruction can be stored.

**Locked**

Pertaining to routines in memory that presently cannot be swapped or transferred.

**Logical device name**

An alphanumeric name assigned by the user to represent a physical device. The name can then be used synonymously with the physical device name in all references to the device. Logical device names are used in device-independent systems to enable a program to refer to a logical device name assigned to a physical device at run-time.

**Loop**

A sequence of instructions that is executed repeatedly until a terminal condition prevails.

**Low-order byte**

The least significant byte in a word. The low-order byte occupies bit positions 0 through 7 in a PDP-11 word and is always an even address.

**Machine language**

The language used by the computer when performing operations.

**Macro**

An instruction in a source language that is equivalent to a specified sequence of assembler instructions, or a com-

mand in a command language that is equivalent to a specified sequence of commands.

**Main program**

The module of a program that contains the instructions at which program execution begins. The main program usually exercises primary control over the operations performed; it also calls subroutines or subprograms to perform specific functions.

**Mask**

A combination of bits that is used to manipulate selected portions of any word, character, byte, or register while retaining other parts for use.

**Mass storage**

Pertaining to a device that can store large amounts of data that are readily accessible to the computer.

**Matrix**

A rectangular array of elements. Any matrix can be considered an array.

**Memory**

Any form of data storage, including main memory and mass storage, in which data can be read and written. Memory usually refers to main memory.

**Memory image**

A replication of the contents of a portion of memory, usually in a file.

**Mnemonic**

An alphabetic easy-to-remember representation of a function or machine instruction.

**Monitor**

The master control program that observes, supervises, controls or verifies the operation of a computer system. The collection of routines that controls the operation of user and system programs, schedules operations, allocates resources, performs I/O, and so forth.

**Monitor command**

An instruction or command issued directly to a monitor from a user.

**Monitor command mode**

The state of the operating system — indicated by a period at the left margin — that allows monitor commands to be entered from the terminal.

**Mount a volume**

To logically associate a physical mass storage medium with a physical device unit. To place a volume on a physical device unit; for example, to place a magtape on a magtape drive and put the drive on line.

**Multiprocessing**

Simultaneous execution of two or more computer programs by a computer which contains more than one central processor.

**Multiprogramming**

A processing method in which more than one task is in an executable state at any one time, even with one CPU.

**Nondirectory-structured**

Refers to a storage volume that is sequential in structure and therefore has no volume directory at its beginning. File information (file name, file type, length, and date-of-creation) is provided with each file on the volume. Such volumes include magtape and cassette.

**Non-file-structured device**

A device, such as a line printer or terminal, in which data cannot be organized as multiple files.

**Object code**

Relocatable machine language code.

**Object module**

The primary output of an assembler or compiler, which can be linked with other object 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 symbols within the module.

**Object Time System (OTS)**

The collection of modules that is called by compiled code in order to perform various utility or supervisory operations; for example, FORTRAN IV Object Time System.

**Octal**

Pertaining to the number system with a radix of eight; for example, octal 100 is decimal 64.

**ODT**

On-line Debugging Technique: an interactive program for finding and correcting errors in programs.

**Off-line**

Pertaining to equipment or devices not currently under direct control of the computer.

**Offset**

The difference between a base location and the location of an element related to the base location. The number of locations relative to the base of an array, string, or block.

**One's complement**

A number formed by interchanging the bit polarities in a binary number; for example, 1s become 0s; 0s become 1s.

**On-line**

Pertaining to equipment or devices directly connected to and under control of the computer.

**Op-code (operation code)**

The part of a machine language instruction that identifies the operation the CPU is to perform.

**Operand**

The data that an instruction operates upon. An operand is usually identified by an address part of an instruction.

**Operating system**

The collection of programs, including a monitor and system programs, that organizes a central processor and peripheral devices into a working unit for the development and execution of application programs.

**Operation**

The act specified by a single computer instruction. A program step undertaken or executed by a computer; for example, addition, multiplication, comparison. The operation is usually specified by the operator part of an instruction.

**Operation code**

See Op-code.

**Operator's console**

The set of switches and display lights used by an operator or a programmer to determine the status of the computer system and to start the computer.

**Option**

An element of a command or command string that enables the user to select alternatives associated with the command. In the RT-11 operating system, an option consists of a slash character (/) followed by the option name and, optionally, a colon, and an option value.

**Output**

The result of a process; the transferring of data from internal storage to external storage.

**Overflow**

A condition that occurs when a mathematical operation yields a result whose magnitude is larger than the hardware is capable of handling.

**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 resident overlay segment.

**Overlay structure**

A program overlay system consisting of a root segment and optionally one or more overlay segments.

**Page**

That portion of a text file delimited by form feed characters and generally 50 to 60 lines long. Corresponds approximately to a physical page of a program listing.

**Parameter**

A variable that is given a constant value for a specific purpose or process.

**Parity**

A binary digit appended to an array of binary digits to make the sum of all bits always odd or always even. It is used to check the validity of data.

**Patch**

To modify a routine in a rough or expedient way, usually by modifying the binary code rather than by assembling it again.

**PC**

See Program counter.

**PDP**

Programmable data processor.

**Peripheral device**

Any device distinct from the computer that can provide input and/or accept output from the computer.

**Physical device**

An I/O or peripheral storage device connected to or associated with a computer.

**Priority**

A number, associated with a task, that determines the order in which the monitor will process the request for service by that task, relative to other tasks requesting service.

**Process**

A set of related procedures and data that are executed and manipulated by a computer.

**Processor**

In hardware, a data processor. In software, a computer program that includes the compiler, assembler, translator, and related functions for a specific programming language (for example, FORTRAN IV processor).

**Processor status word (PSW)**

A register in the PDP-11 that indicates the current priority of the processor, the condition of the previous operation, and other basic control items.

**Program**

A set of machine instructions or symbolic statements combined to perform some task.

**Program counter (PC)**

A register used by the central processor unit to record the addresses of the instructions to be executed. The PC (register 7 of the eight general registers) always contains the address of the next instruction to be executed, or the second or third word of the current instruction.

**Program development**

The process of writing, entering, translating, and debugging source programs.

**Programmed request**

A set of instructions (available only to programs) that is used to invoke a monitor service.

**Program section**

A named, contiguous unit of code (instructions or data) that is considered as an entity and that can be relocated separately without destroying the logic of the program.

**Protocol**

A formal set of conventions governing the format and relative timing of information exchange between two communicating processes.

**PSW**

See Processor status word.

**Queue**

Any dynamic list of items; for example, items waiting to be scheduled or processed according to system- or user-assigned priorities.

**Radix**

The base of a number system; the number of digit symbols required by a number system.

**RAM (Random-Access Memory)**

Memory that is accessed in such a way that the next location from which data is to be obtained is not dependent on the location of the previously obtained data.

**Random access**

Access to data in which the next location from which data is to be obtained is not dependent on the location of the previously obtained data. Contrast Sequential access.

**Read-only memory (ROM)**

Memory whose contents are not alterable by computer instructions.

**Real-time processing**

The computation performed while a related or controlled physical activity is occurring. The results of the computation can be used for guiding the process.

**Record**

A collection of related items of data treated as a unit; for example, a line of source code or a person's name, rank, and serial number.

**Recursive**

Pertaining to a repetitive process in which the result of each process is dependent upon the result of the previous one.

**Re-entrant**

Pertaining to a program composed of a shareable segment of pure code and a nonshareable segment that is the data area.

**Register**

See General register.

**Relative address**

The number that specifies the difference between the actual address and a base address.

**Relocate**

In programming, to move a routine from one portion of storage to another and to adjust the necessary address references so that the routine, in its new location, can be executed.

**Resident**

Pertaining to data or instructions that are permanently located in main memory.

**Resource**

Any means available to users, such as computational power, programs, data files, storage capacity, or a combination of these.

**Restart**

To resume execution of a program.

**ROM**

See Read-only memory.

**Root segment**

The segment of an overlay structure that, when loaded, remains resident in memory during the execution of a program.

**Routine**

A set of instructions arranged in proper sequence to cause a computer to perform a desired operation.

**Run**

A single, continuous execution of a program.

**Sector**

A physical portion of a mass storage device.

**Segment**

See Overlay segment.

**Sequential access**

A method of data access in which the next location from which data is to be obtained immediately follows the location of the previously obtained data. Contrast Random access.

**Software**

The collection of programs and routines associated with a computer. Compilers and library routines are examples.

**Software bootstrap**

A bootstrap that is activated by loading the instructions of the bootstrap and specifying the appropriate load and start address.

**Source code**

Text, usually in the form of an ASCII format file, that represents a program. Such a file can be processed by an appropriate system program.

**Source language**

The system of symbols and syntax used to describe a procedure that a computer can execute.

**Spooling**

The technique by which I/O with slow devices is placed on mass storage devices to await processing.

**Storage**

Pertaining to a device into which data can be entered, in which it can be held, and from which it can be retrieved at a later time.

**String**

A connected sequence of entities, such as a line of characters.

**Subprogram**

A program or a sequence of instructions that can be called to perform the same task (though perhaps on different data) at different points in a program, or in different programs.

**Subroutine**

See Subprogram.

**Subscript**

A numeric valued expression or expression element that is appended to a variable name to uniquely identify specific elements of an array. Subscripts are enclosed in parentheses. There is a subscript for each dimension of an array. Multiple subscripts must be separated by commas. For example, a two-dimensional subscript might be (2,5).

**Supervisory programs**

Computer programs that have the primary function of scheduling, allocating, and controlling system resources.

**Swapping**

The process of moving data from memory to a mass storage device, temporarily using the empty memory area for another purpose, and then restoring the original data to memory.

**Synchronous**

Pertaining to related events where all changes occur simultaneously or in definite timed intervals.

**Syntax**

The structure of expressions in a language and the rules governing the structure of a language.

**System program**

A program that performs system-level functions. A program that is part of the basic operating system (for example, a system utility program) is a system program.

**System volume**

The volume on which the operating system is stored.

**Table**

A collection of data in a well-defined list.

**Terminal**

An I/O device, such as an LA120 terminal, that includes a keyboard and a display mechanism. In PDP-11 systems, a terminal is used as the primary communication device between a computer system and a user.

**Time sharing**

A method of allocating resources to multiple users so that the computer processes a number of programs concurrently.

**Toggle**

To use switches on the computer operator's console to enter data into the computer memory.

**Translate**

To convert from one language to another.

**Trap**

A conditional jump to a known memory location performed automatically by hardware as a side effect of executing a processor instruction. The address location from

which the jump occurs is recorded. It is distinguished from an interrupt, which is caused by an external event.

**Truncation**

The reduction of precision by ignoring one or more of the least significant digits; for example, 3.141597 truncated to four decimal digits is 3.141.

**Turnkey**

Pertaining to a computer system sold in a ready-to-use state.

**Two's complement**

A number used to represent the negative of a given value in many computers. This number is formed from the given binary value by changing all 1s to 0s and all 0s to 1s and then adding 1.

**Underflow**

A condition that occurs when a mathematical operation yields a result whose magnitude is smaller than the smallest amount the hardware can handle.

**User program**

An application program.

**Utility program**

Any general-purpose program included in an operating system to perform common functions.

**Variable**

The symbolic representation of a logical storage location that can contain a value that changes during a processing operation.

**Vector**

A consecutive list of associated data.

**Volume**

A mass storage medium that can be used for file-structured data storage.

**Wildcard**

A valid substitute for characters in a file specification. Used to perform operations on multiple files. Can be asterisks to represent entire file names or file types, or percent signs to represent single characters in file names or file types.

**Wildcard operation**

A shorthand method of referring to all files with a specific characteristic in their name.

**Word**

Sixteen binary digits treated as a unit in PDP-11 computer memory.

**Write-enabled**

The condition of a volume that allows information to be written on it.

**Write-protected**

The condition of a volume that protects the volume against information being written on it.

# INDEX

- Absolute program sections, 12-4
- Addresses
  - assignment by LINK, 12-3
- Advance (A) command (EDIT), 5-9
- Application packages, 1-10
- Assembler, 11-2
- Assembler errors, 11-7
- Assembly language
  - See Machine-level language
- Assembly listings, 11-8
- ASSIGN keyboard command
  - assigning logical device names, 4-11
  - changing the output device, 9-11, 15-5
- Background job
  - creating, 15-2
  - directing input to, 15-4
  - editing, 15-2
  - executing, 15-5
  - running, 15-2
  - terminating, 15-6
- Background program
  - running, 15-1
- Backup copy
  - files, 17-1
  - system volume, B-3
- BASIC-11
  - commands, 10-3 to 10-13
  - command summary
    - edit, 10-6
    - execution, 10-11
    - file maintenance, 10-13
    - demonstration program, 10-7
    - errors, 10-9
    - running, 10-1
  - creating, 10-4, 10-12
  - editing, 10-4
  - maintaining files, 10-12
  - replacing, 10-13
  - running, 10-8
  - saving, 10-12
  - using, 10-12
- BASIC-11 programming language, 1-10, 8-3
- BASIC keyboard command, 10-2
- BATCH
  - stream, 17-1
- Beginning (B) command (EDIT), 5-4
- Bit
  - definition of, 11-5
- Bootstrap
  - manual operations, A-1
  - procedure, 2-4
  - prompts and responses (table), 2-6
  - pushbutton console, A-1, A-4
  - relationship with computer (figure), 2-2
  - suggestions for bootstrapping, B-2

Bootstrap (cont.)  
   switch register console, A-1, A-5  
   terminal keyboard, A-3  
   typing, A-3  
 Breakpoints  
   clearing, 14-9  
   clearing all, 14-10  
   setting, 14-8  
 /BRIEF  
   DIRECTORY option, 4-14  
 BYE command  
   BASIC-11, 10-3  
 Byte  
   definition of, 11-5  
  
 Character insertion  
   immediate mode, 5-17  
 Character search (EDIT), 5-9  
 Command arguments (EDIT)  
   table of, 5-5  
 Commands  
   BASIC  
     See BASIC-11  
   control  
     format of, 4-3  
   correcting typing errors, 4-4  
   EDIT  
     See EDIT  
   keyboard  
     See Keyboard commands  
     format of, 4-2  
   ODT  
     See ODT  
 Compiler, 9-2  
 Computer  
   hardware configuration, 2-3  
   memory, 2-1  
   PDP-11 (figure), 1-3  
 Console, 1-4, 3-1  
   See also Terminals  
 COPY keyboard command, 7-3  
 /CREATE  
   EDIT option, 5-2  
   LIBRARY option, 13-2, 13-5  
 CREF table, 11-10  
 /CROSSREFERENCE  
   MACRO option, 11-7  
 Cross-reference (CREF) listing, 11-10  
 Cross-reference (CREF) table  
   See CREF table  
 CTRL/B, 15-4  
 CTRL/C CTRL/C  
   aborting program execution, 4-14  
   returning to BASIC-11 command mode,  
     10-9  
   terminating background job, 15-6  
   terminating indirect file execution,  
     16-4  
 CTRL/C ESCAPE ESCAPE (EDIT), 5-6,  
   5-19  
 CTRL/D, 5-19  
 CTRL/E, 4-7  
 CTRL/F, 15-4  
 CTRL/G, 5-18  
 CTRL/L, 5-10  
 CTRL/N, 5-18  
 CTRL/O, 4-12  
 CTRL/U, 4-4  
 CTRL/U (EDIT), 5-3, 5-8  
 CTRL/V, 5-18  
 CTRL/X, 5-8  
 CTRL key, 3-4  
  
 Date  
   See also Time  
   displaying, 4-9  
   entering, 4-8  
 DATE keyboard command, 4-8  
 /DEBUG  
   LINK option, 14-5  
 Debugging a program, 1-9  
   See also ODT  
   techniques, 14-2  
 Decimal/octal/binary conversion, 11-6  
 Delete (D) command (EDIT), 5-8  
 DELETE (DEL) command  
   BASIC-11, 10-5  
 DELETE key  
   correcting typing errors, 4-4  
   editing, 5-3, 5-8  
   function, 3-3  
   immediate mode, 5-18  
   ODT, 14-6  
 DELETE keyboard command, 7-6  
 Demonstration programs  
   BASIC-11, 10-7  
   creating, 5-19  
   FORTRAN, 5-20  
   library files, 13-3  
   load maps, 12-8  
   MACRO, 5-21  
 Device assignments  
   changing, 9-11  
 Device handlers, 1-8  
 Device names, 4-9  
 Devices  
   random-access, 1-5  
   random-access (figure), 1-5  
 Device unit, 3-5, 4-10

**DIFFERENCES** keyboard command, 6-2  
**DIRECTORY** keyboard command, 4-12  
 Directory listings, 3-7, 4-12  
     generating, 7-1  
 Directory-structured volumes, B-4  
 Documentation, 1-10  
     hardware manuals, 1-10  
     software manuals, 1-11  
     source listings, 1-11  
**Drive**  
     See Device unit  
**DUMP**, 1-9  
**EDIT**  
     command arguments (table), 5-5  
     commands  
         summary of, 5-13  
     exiting, 5-19  
**EDIT** commands, 5-3 to 5-13  
**Editing** command mode, 5-3  
**Editing** commands  
     **ESCAPE**  
         returning to editing command mode,  
         5-19  
**Editing** commands, multiple  
     entering, 5-8  
     erasing, 5-8  
**EDIT** keyboard command  
     creating a file, 5-2  
     editing a file, 5-4  
**Edit** lower (EL) command (EDIT), 5-12  
**Editors**  
     **EDIT**  
         See EDIT  
**Edit** upper (EU) command (EDIT), 5-13  
**Errors**  
     avoiding programming, 14-1  
     types of  
         assembler, 11-7  
         clerical, 14-2  
         compiler, 9-4  
         logical, 14-2  
         syntax, 14-2  
**ESC**  
     See ESCAPE key  
**ESCAPE** command  
     entering multiple commands, 5-8  
     returning to editing command mode,  
     5-19  
**ESCAPE** ESCAPE command  
     activating immediate mode, 5-17  
     executing editing commands, 5-3  
     executing multiple editing commands,  
     5-8  
**ESCAPE** key, 3-4, 5-3  
     See also ESCAPE command and  
     ESCAPE ESCAPE command  
**Examples**  
     re-creating, 4-3  
**EXECUTE** keyboard command, 9-12,  
     11-16  
**Exit** (EX) command (EDIT), 5-3  
**FB** monitor, 15-3  
**File** maintenance  
     BASIC-11, 10-12  
     programs, 1-9  
     See also File maintenance commands  
**File** maintenance commands  
     summary, 7-9  
**File** maintenance operations, 7-1  
**File** names, 3-7  
     changing, 7-5  
**Files**  
     backup copy, 5-12, 17-1  
     closing, 5-3, 5-11  
     comparing, 6-1, 6-2  
     copying, 7-3  
     creating, 5-2  
     deleting, 7-5  
     editing, 5-4  
     editing (figure), 5-2  
     indirect  
         See Indirect command files  
     paging, 5-2  
     protecting, 3-7, 7-7  
     removing protection from, 7-7  
     renaming, 5-6, 7-4  
     storing, 3-7  
     transferring, 7-3  
**File** types, 3-7  
     changing, 7-5  
**File** types (table), 4-13  
**Foreground/background** environment,  
     15-1  
**Foreground/background** monitor  
     See FB monitor  
**Foreground/background** program  
     communication, 15-3  
**/FOREGROUND/LINK** option, 15-4  
**Foreground** job  
     creating, 15-4  
     directing input to, 15-4  
     executing, 15-5  
     linking, 15-4  
     loading device handlers, 15-5  
     terminating, 15-7  
     unloading, 15-7

Foreground program  
   running, 15-1  
 FORLIB.OBJ, 9-3, 12-7  
 FORTRAN  
   compiler, 9-2  
   demonstration program, 5-20  
     running, 9-1  
   library modules, 9-2  
   object time system (OTS), 9-2  
   programming language, 1-10, 8-3  
 FORTRAN/BASIC language volume, B-5  
 FORTRAN demonstration program  
   errors, 9-7  
 FORTRAN keyboard command, 9-4  
 FORTRAN language processor, 9-1  
 FORTRAN program  
   compiling, 9-3  
   execution commands  
     summary, 9-14  
   linking, 9-8, 9-9, 12-7  
   producing a load map, 12-7  
   producing a load module, 12-7  
   running, 9-11  
   sectioning, 12-5  
 FORTRAN programming language, 9-1  
 FRUN keyboard command, 15-6  
  
 Get (G) command (EDIT), 5-9  
 Global symbols, 12-2  
 Graphics display terminal  
   See VT11 display hardware  
 GT keyboard command, 4-6  
 GT OFF keyboard command, 4-7, 5-2  
 GT ON keyboard command, 4-6, 5-16  
  
 Hardware, 1-1  
   computer, 1-1  
   storage medium, 1-4  
   terminal, 1-3  
 Hardware configuration, 2-1  
   computer, 2-3  
   languages, 2-4  
   optional devices, 2-4  
   storage volume, 2-4  
   system volume, 2-3  
   terminal, 2-3  
 Hardware manuals, 1-10  
 HELP file, 17-2  
 HELP keyboard command, 17-2  
 High-level languages, 1-10, 8-1  
   See also BASIC-11 and FORTRAN  
  
 Immediate mode  
   BASIC-11, 10-3  
  
 Immediate mode (EDIT)  
   character insertion, 5-17  
   VT11 display hardware, 5-16  
 Immediate mode (EDIT) commands,  
   5-17 to 5-19  
   VT11 display hardware (table), 5-17  
 Indirect command files, 16-1, 17-1  
   creating, 16-1  
   entering monitor commands, 16-1  
   executing, 16-4  
   using, 16-1  
   using the editor to create, 16-2  
 INITIALIZE keyboard command, 4-15  
 Initializing volumes, 4-15  
 Input/output devices  
   See Peripheral devices  
 /INSERT  
   LIBRARY option, 13-6  
 Insert (I) command (EDIT), 5-3  
 Internal symbols, 12-2  
 Interpreter  
   description of, 10-2  
  
 Jobs  
   background, 15-2  
   foreground, 15-4  
 Jump (J) command (EDIT), 5-7  
  
 Keyboard commands, 4-1  
 Keyboard layouts (figure), 3-3  
 Keyboard monitor  
   See KMON  
 Keyboard symbols (table), 4-4  
 Kill (K) command (EDIT), 5-9  
 KMON, 4-1  
  
 Language comparisons (table), 8-2  
 Language processors, 1-10, 8-1  
   BASIC-11, 10-1  
   FORTRAN, 9-1  
   MACRO, 11-2  
 Languages  
   See Programming languages  
 Language volume  
   FORTRAN/BASIC, B-5  
 Librarian, 1-9  
   See also Library files  
 Library files  
   creating, 13-2  
   demonstration programs, 13-3  
   macro, 13-1  
   maintaining, 13-2  
   maintenance commands for  
     summary, 13-7

- Library files (cont.)
  - object libraries, 13-1
- LIBRARY keyboard command, 13-2, 13-5
- Library modules, 9-2
- Library references
  - resolving, 12-2
- LINE FEED key, 3-4
  - ODT, 14-7
- Linking a program, 1-9
  - See also LINK keyboard command and Link operation
- LINK keyboard command, 9-9
  - linking a foreground program, 15-4
  - linking a MACRO program, 11-14
  - linking ODT, 14-5
- Link operations, 12-1
  - address assignment, 12-3
  - overlay feature, 12-6
  - producing a load map, 12-7
  - producing a load module, 12-7
  - program relocation, 12-3
  - program sections, 12-4
  - resolving library references, 12-2
  - resolving symbolic references, 12-2
  - summary of commands, 12-11
- Link volume, B-8
- /LIST
  - FORTTRAN option, 9-4
  - LIBRARY option, 13-6
  - MACRO option, 11-7
- List (L) command (EDIT), 5-5
- LIST command
  - BASIC-11, 10-5
- LISTNH command
  - BASIC-11, 10-6
- LOAD keyboard command, 15-5
- Load maps
  - demonstration programs, 12-8
  - producing, 12-7
- Load modules
  - producing, 12-7
- Logical device names
  - assigning, 4-9
  - special (table), 4-10
- Lowercase characters
  - EDIT, 5-12
- Machine language code, 11-4
- Machine-level language, 1-10, 8-1
  - See also MACRO
- /MACRO
  - LIBRARY option, 13-2
- MACRO
  - assembly listing, 11-8
  - demonstration program, 5-21
  - errors, 11-12
  - running, 11-1
  - programming language, 8-3, 11-1
- MACRO assembler, 11-2
  - running, 11-15
- MACRO keyboard command, 11-7
- MACRO language processor, 11-2
- Macro library files, 13-1
- MACRO programs
  - assembling, 11-6
  - developing, 11-1
  - linking, 11-13, 11-14, 12-7
  - producing a load map, 12-7
  - producing a load module, 12-7
  - sectioning, 12-5
  - summary of execution commands, 11-17
- Macros, 11-11
- /MAP
  - LINK option, 12-7
- /MATCH
  - DIFFERENCES option, 6-3
- Memory, 2-1, 11-3
- Memory image load module, 11-14
- Monitor
  - description of, 1-8
- Monitor command format, 4-2
- Monitor command language
  - See Keyboard commands
- Monitor program, 3-1
- NEW command
  - BASIC-11, 10-12
- Next (N) command (EDIT), 5-11
- Nondirectory-structured volumes, B-4
- Object libraries, 13-1
  - building, 13-2
  - creating input files, 13-2
  - listing, 13-6
  - updating, 13-6
- Object module relocation (figure), 12-4
- Object modules, 12-1
  - linking
    - FORTTRAN, 9-8
    - MACRO, 11-13
  - linking (figure), 9-9
- Object programs
  - linking, 12-1
- Object time system (OTS), 9-2
- ODT, 14-3
  - accessing general registers, 14-9
  - clearing breakpoints, 14-9

ODT (cont.)  
 closing the currently open location  
 address, 14-7  
 commands  
 summary, 14-12  
 continuing execution, 14-9  
 executing MACRO programs, 14-7  
 linking with a program, 14-5  
 location addresses, 14-7  
 opening addressed locations, 14-9  
 opening bytes, 14-10  
 opening location addresses, 14-7  
 opening sequential location addresses,  
 14-7  
 relocation registers, 14-7  
 removing a breakpoint, 14-9  
 removing all breakpoints, 14-10  
 running, 14-6  
 running a program with, 14-8  
 setting breakpoints, 14-8  
 setting relocation registers, 14-7  
 ODT commands, 14-6 to 14-10  
 OLD command  
 BASIC-11, 10-12  
 On-line debugging technique  
 See ODT  
 Operating system  
 applications packages, 1-10  
 description of, 1-8  
 device handlers, 1-8  
 language processors, 1-10  
 monitor program, 1-8  
 utility programs, 1-8  
 Operating system (figure), 1-9  
 Optional devices, 1-6  
 hardware configuration, 2-4  
 Output device  
 changing, 9-11, 15-5  
 Overlay feature  
 See Overlay segments  
 Overlay segments, 12-6  
  
 Peripheral devices, 1-6  
 Peripheral devices (figure), 1-6  
 Physical device names (table), 4-9  
 /PRINTER  
 DIRECTORY option, 4-14  
 Printer  
 enabling, 4-7  
 PRINT keyboard command, 7-8  
 Processor  
 stopping the, B-1  
 Program counter, 11-3  
 Programmed requests, 11-11  
 Programming languages, 8-1  
 BASIC-11, 8-3, 10-1  
 choosing, 8-1  
 comparing (table), 8-2  
 DIBOL, 8-3  
 FORTRAN, 8-3, 9-1  
 hardware configuration, 2-4  
 MACRO, 8-3, 11-1  
 Program relocation, 12-3  
 Programs  
 See also FORTRAN program,  
 BASIC-11 program, and MACRO  
 programs  
 debugging, 14-1  
 Program sections  
 absolute, 12-4  
 blank, 12-5  
 instruction, 12-5  
 named relocatable, 12-4  
 /PROMPT  
 LINK option, 12-6  
 Prompts  
 bootstrap, 2-6  
 EDIT, 5-3  
 monitor, 4-1  
 PROTECT keyboard command, 7-7  
 Pushbutton console  
 using to bootstrap, A-4  
 Pushbutton console (figure), A-4  
  
 /QUERY  
 DELETE option, 7-6  
  
 Radix  
 conversion table, 11-6  
 Random-access devices  
 See Devices  
 Read (R) command (EDIT), 5-4  
 Relocatable program sections, 12-4  
 Relocation registers  
 ODT, 14-7  
 /REMOVE  
 LIBRARY option, 13-7  
 RENAME keyboard command, 5-6, 7-5  
 Renaming files  
 cassette users, B-4  
 magtape users, B-4  
 REPLACE command  
 BASIC-11, 10-13  
 Resident monitor  
 See RMON  
 RETURN key  
 executing commands, 4-1, 4-3  
 function, 3-4  
 ODT, 14-7

- RMON, 4-1
- RT-11 computer system
  - description of, 1-1
- RT-11 computer system (figure), 1-2
- RT-11 operating system
  - See Operating system
- RUN command
  - BASIC-11, 10-8
  - ODT, 14-8
- RUN keyboard command
  - background job, 15-3
- RUNNH command
  - BASIC-11, 10-8
  
- SAVE command
  - BASIC-11, 10-12
- SCRATCH (SCR) command
  - BASIC-11, 10-6
- SHIFT key, 3-3
- SHOW keyboard commands, 4-11
- Software
  - defined, 1-7
  - operating system, 1-8
- Software (figure), 1-8
- Software manuals, 1-11
- Source comparison program, 1-9
  - See also Source comparison
- Source files
  - comparing, 6-1
- Source listings, 1-11
- Storage medium
  - definition of, 1-4
  - random-access
    - See Devices
- Storage volumes
  - hardware configuration, 2-4
  - initializing, 4-15
  - loading, 2-5
  - protecting files, 3-7
  - using, 3-4, 3-7
- Storage volumes (figure), 3-6
- SUBSTITUTE (SUB) command
  - BASIC-11, 10-4
- Switch register console
  - using to bootstrap, A-5
- Switch register console (figure), A-5
- Symbolic references
  - resolving, 12-2
- Symbols
  - global, 12-2
  - internal, 12-2
- Symbol table, 11-4, 11-10
- SYSLIB.OBJ, 9-2, 12-7
- System macro library, 11-11
- System volume, 2-2
  - backing up, B-3
  - hardware configuration, 2-3
  - loading, 2-5
- System volume (table), 2-4
  
- TAB key, 3-4
- Terminal
  - hardware configuration, 2-3
- Terminal (figure), 3-2
- Terminal devices (figure), 1-4
- Terminals, 1-3, 3-1
  - console, 1-4
- Text buffer, 5-1
  - pointer, 5-4
- Time
  - See also Date
  - displaying, 4-9
  - entering, 4-8
- TIME keyboard command, 4-8
- TYPE keyboard command, 7-8
  
- UNLOAD keyboard command, 15-7
- UNPROTECT keyboard command, 7-7
- User service routine
  - See USR
- USR, 4-1
- Utility programs, 1-8
  
- Verify (V) command (EDIT), 5-7
- Volume directory
  - file storage, 3-7
  - listing, 4-12
  - operations, 7-1
- Volume structures
  - comparing, B-4
- Volume substitution during operations, B-6
- VT11 display hardware
  - commands, 4-6, 4-7
  - enabling, 4-6
  - enabling the printer, 4-7
  - immediate mode, 5-16
  - using, 4-5, 5-15
- VT11 display hardware (figure), 4-6
  
- Wildcards
  - using
    - with DELETE keyboard command, 7-6
    - with DIRECTORY keyboard command, 5-12
- Word
  - definition of, 11-5
- Write enable
  - file protection, 3-7
- Write protect
  - file protection, 3-7



# HOW TO ORDER ADDITIONAL DOCUMENTATION

From	Call	Write
Chicago	312-640-5612 8:15 A.M. to 5:00 P.M. CT	Digital Equipment Corporation Accessories & Supplies Center 1050 East Remington Road Schaumburg, IL 60195
San Francisco	408-734-4915 8:15 A.M. to 5:00 P.M. PT	Digital Equipment Corporation Accessories & Supplies Center 632 Caribbean Drive Sunnyvale, CA 94086
Alaska, Hawaii	603-884-6660 8:30 A.M. to 6:00 P.M. ET  or 408-734-4915 8:15 A.M. to 5:00 P.M. PT	
New Hampshire	603-884-6660 8:30 A.M. to 6:00 P.M. ET	Digital Equipment Corporation Accessories & Supplies Center P.O. Box CS2008 Nashua, NH 03061
Rest of U.S.A., Puerto Rico*	1-800-258-1710 8:30 A.M. to 6:00 P.M. ET	
*Prepaid orders from Puerto Rico must be placed with the local DIGITAL subsidiary (call 809-754-7575)		
Canada		
British Columbia	1-800-267-6146 8:00 A.M. to 5:00 P.M. ET	Digital Equipment of Canada Ltd 940 Belfast Road Ottawa, Ontario K1G 4C2 Attn: A&SG Business Manager
Ottawa-Hull	613-234-7726 8:00 A.M. to 5:00 P.M. ET	
Elsewhere	112-800-267-6146 8:00 A.M. to 5:00 P.M. ET	
Elsewhere		Digital Equipment Corporation A&SG Business Manager*
*c/o DIGITAL's local subsidiary or approved distributor		



**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 \_\_\_\_\_ Telephone \_\_\_\_\_

Street \_\_\_\_\_

City \_\_\_\_\_ State \_\_\_\_\_ Zip Code \_\_\_\_\_  
or Country

Do Not Tear — Fold Here and Tape

**digital**

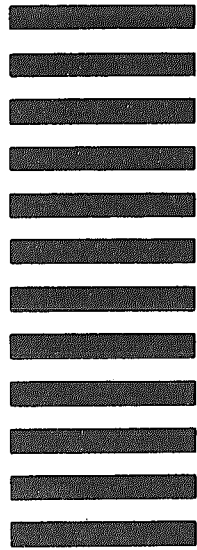


No Postage  
Necessary  
if Mailed in the  
United States

**BUSINESS REPLY MAIL**  
FIRST CLASS PERMIT NO.33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

**SSG/ML PUBLICATIONS, MLO5-5/E45  
DIGITAL EQUIPMENT CORPORATION  
146 MAIN STREET  
MAYNARD, MA 01754-2571**



Do Not Tear — Fold Here

Cut Along Dotted Line