March 1980

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.

Introduction to RT-11

Order No. AA-5281B-TC

SUPERSESSION/UPDATE INFORMATION: This manual supersedes Order No. DEC-11-ORITA-A-D, DN1.

OPERATING SYSTEM AND VERSION: RT-11 V4.0

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

digital equipment corporation · maynard, massachusetts

First Printing, March 1980

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

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

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

The following are trademarks of Digital Equipment Corporation:

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

10/80-15

CONTENTS

FIGURES

TABLES

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. It supports a number of programming languages, including industry-standard FORTRAN and BASIC; APL; and — for more advanced users — the PDP-11 assembly language, MACRO-11. In addition, it provides a comprehensive set of operating commands that programmers at all levels use to control system operations.

MANUAL INTENT

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 does this by first presenting the background material 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 re-create; 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.

MANUAL DESIGN

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

- Those having little or no previous "hands-on" computer experience (including those whose experience has been limited to batch environments)
- Those who are experienced users of a computer system other than RT-11
- 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 4 and later releases)

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.

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 the manual (BASIC-11 and FORTRAN IV capabilities are optional).

If your choice is FORTRAN IV, read Chapter 9, Running a FORTRAN IV Program.

If you wish to use BASIC-11, read Chapter 10, Running a BASIC-11 Program.

If you choose MACRO-11, read Chapter 11, Running an Assembly Language Program.

MACRO and FORTRAN users continue to Chapter 12, Linking Object Programs, and Chapter 13, Constructing Library Files.

All users should read Chapter 14, Debugging a User Program, which provides some suggestions for finding and fixing errors in user programs.

Those users who plan to exercise the foreground/background capability of the RT-11 system should read Chapter 15, Using the Foreground/Background Monitor.

Finally, all users should continue to Chapter 16, Using Indirect Files, which describes the procedure for performing operations unattended, and Chapter 17, which gives some advice to new users, including the use of the RT-11 Help file.

Two appendixes are provided for reference. Appendix A discusses system bootstrapping procedures that are not generally needed, but may be required by some system users. 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 4 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 important 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.

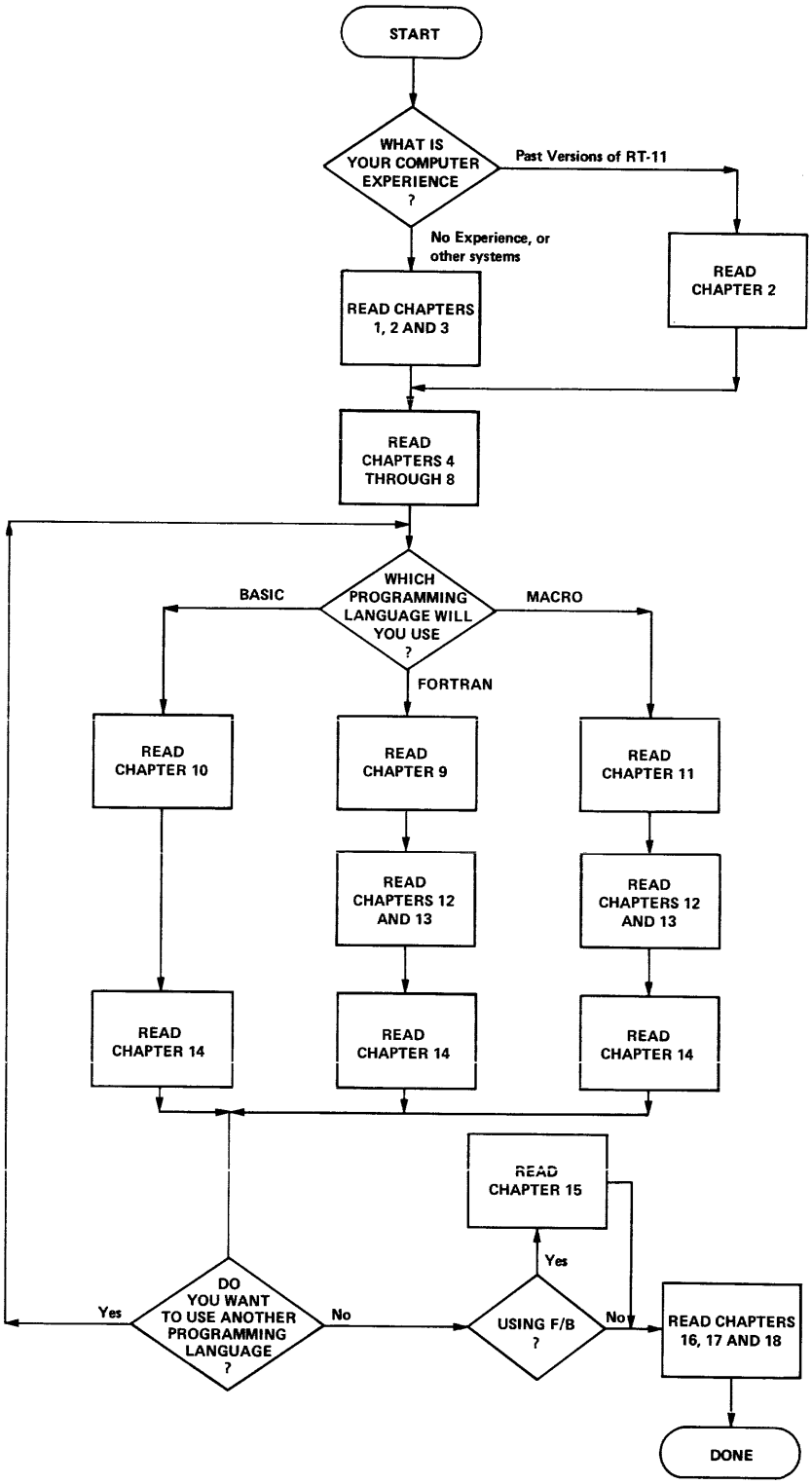


Figure 1 Flowchart for Selective Reading

CHAPTER 1
INTRODUCING THE RT-11 COMPUTER SYSTEM

A computer system is a collection of components working 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. To accomplish this goal, hardware elements are combined with software elements to form a functioning unit. 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 that have been written for the system; these 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.[1]

SYSTEM HARDWARE
SYSTEM SOFTWARE
+SYSTEM DOCUMENTATION
COMPUTER SYSTEM

SYSTEM HARDWARE

The RT-11 computer system requires three basic hardware items: the computer itself, 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.

The Computer

The computer does the real work of the system; it 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.

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 can be 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.

[1] 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

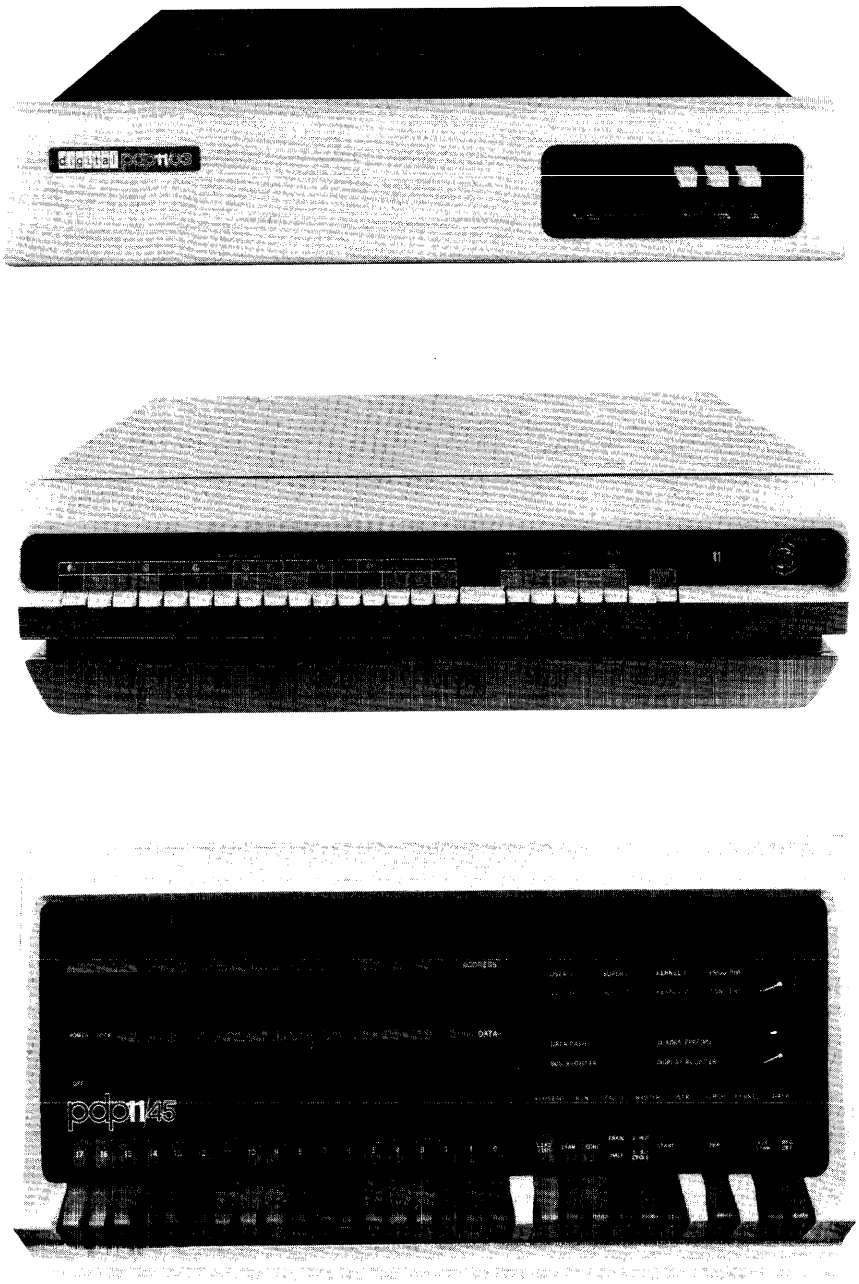
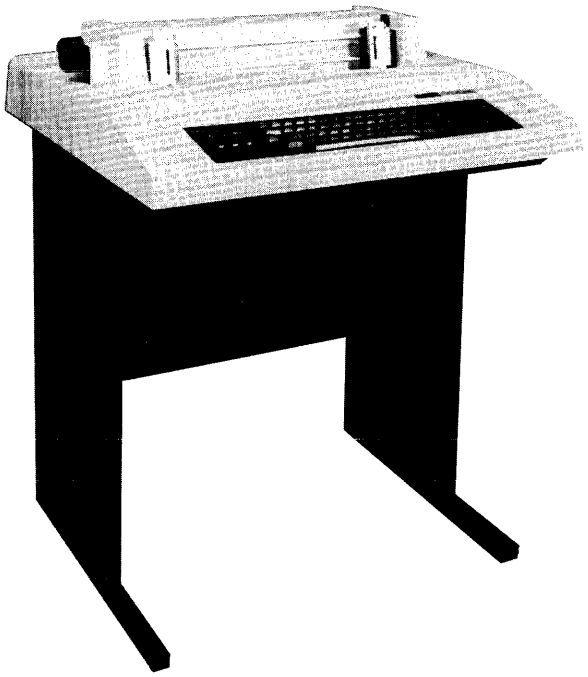


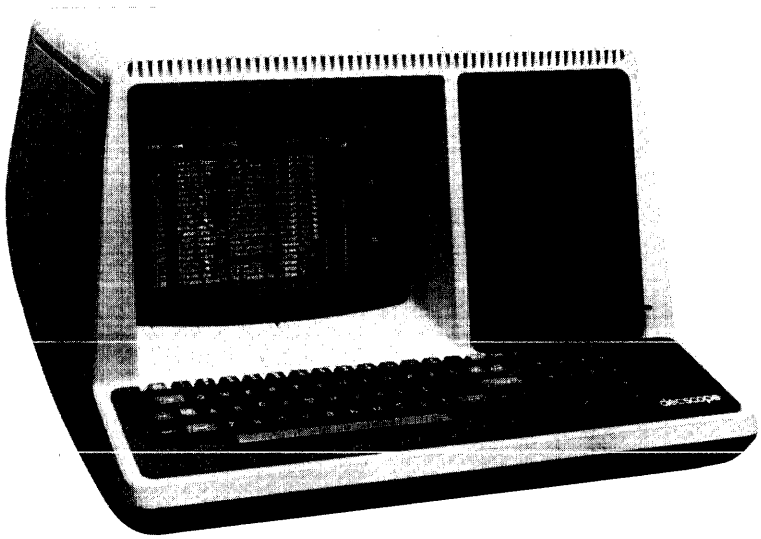
Figure 1-2 PDP-11 Computers

The Terminal

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 many of the terminal devices that can be used in an RT-11 computer system.



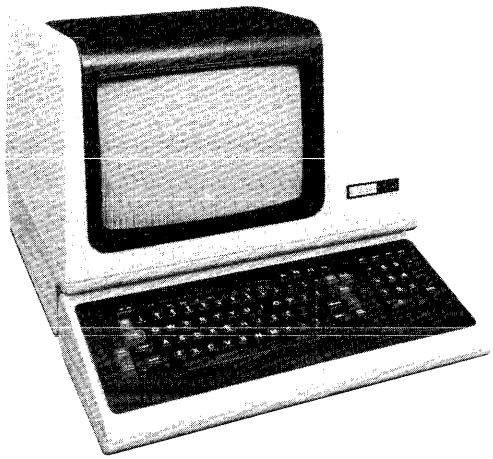
LA36



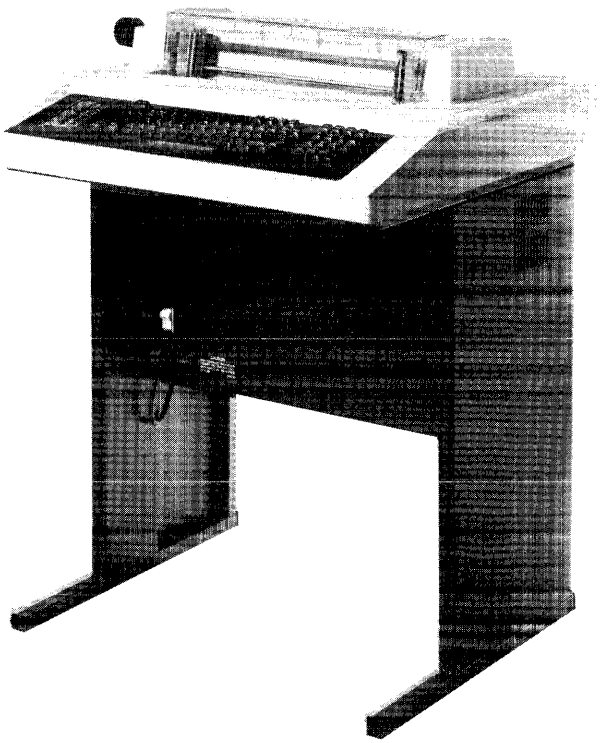
VT52

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.



VT100



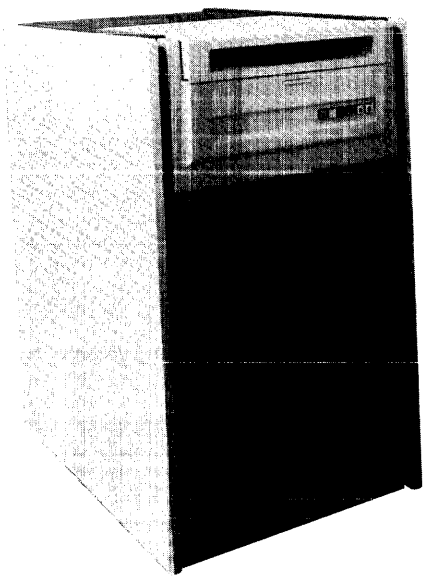
LA120

Figure 1-3 Terminal Devices (Cont.)

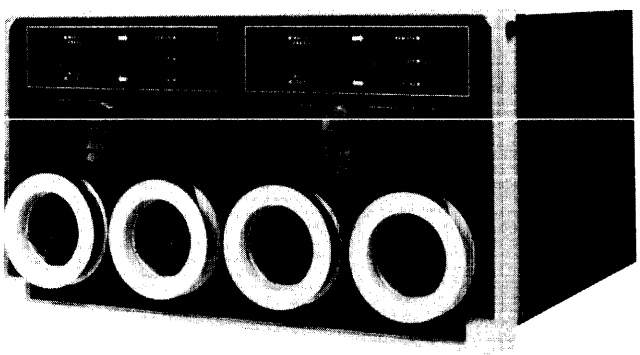
The Storage Medium

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 (within 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.)

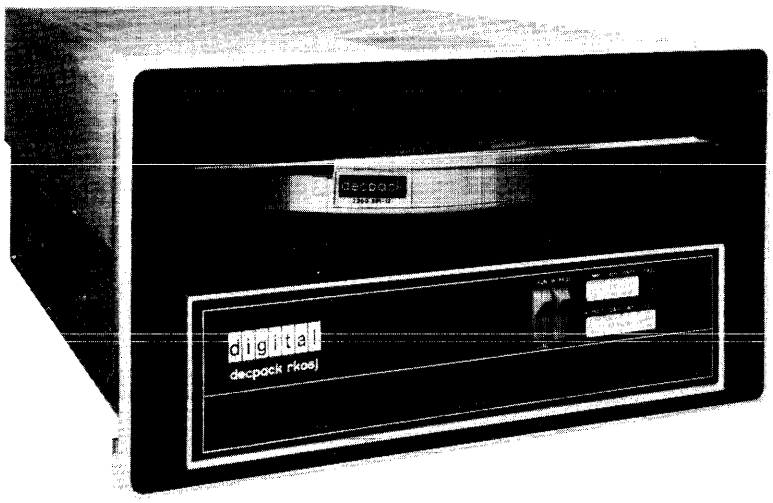


RK06

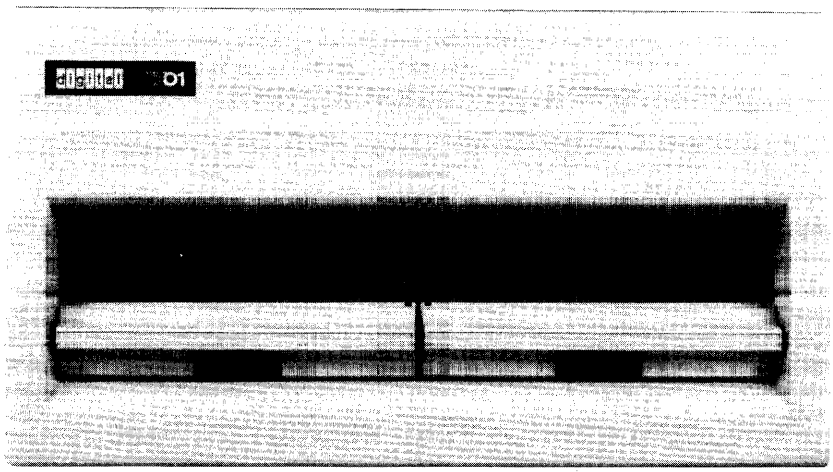


DECtape

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



RK05



RX01 Diskette

Figure 1-4 Random-Access Storage Media and Their Devices (Cont.)

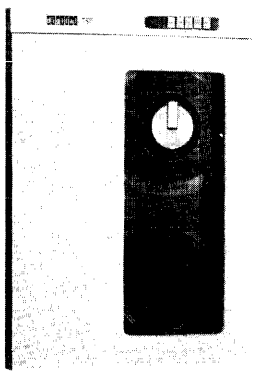
These three devices — the computer, the terminal, and the storage medium — are the required hardware components of an RT-11 computer system. With the exception of the computer, all hardware devices are called peripheral devices. Peripheral devices supplement the computer by providing external resources 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

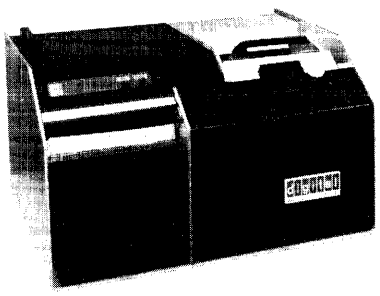
Optional peripheral devices are added to a computer system according to the specific needs of the system users. For example, computer systems that are used primarily for program develop-

ment 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.

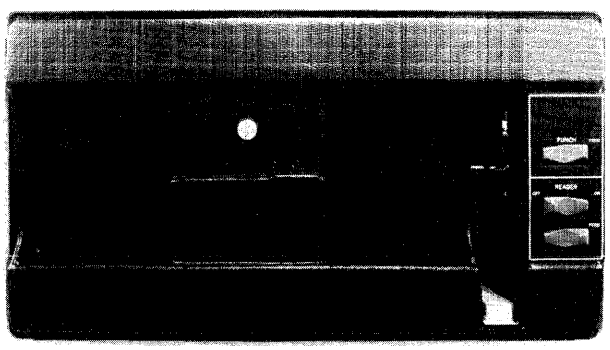
Peripheral devices are categorized as input/output (I/O) devices since the functions they perform provide information (input) to the computer, accept information (output) from the computer, or do both. Some common input devices are card readers, paper tape readers, and programmable clocks. Output devices include line printers, paper tape punches, and plotters. Input/output devices include terminals and storage devices because they are capable of performing 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.



Magtape

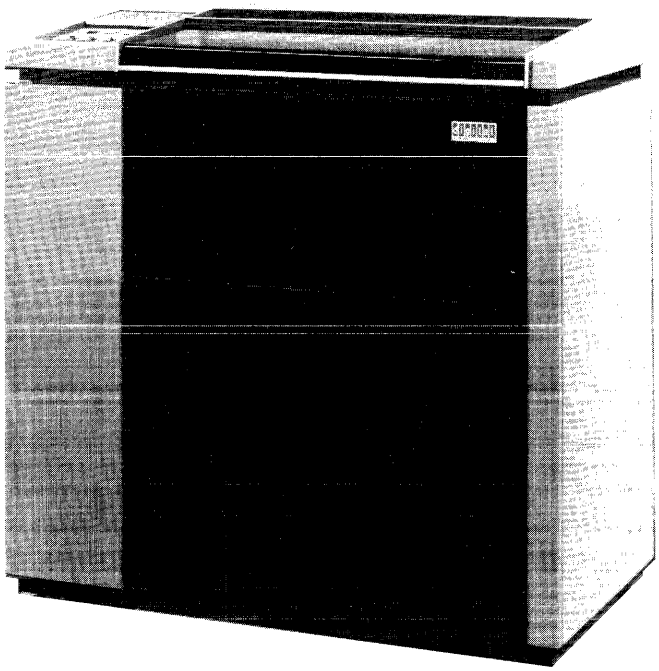


Card Reader

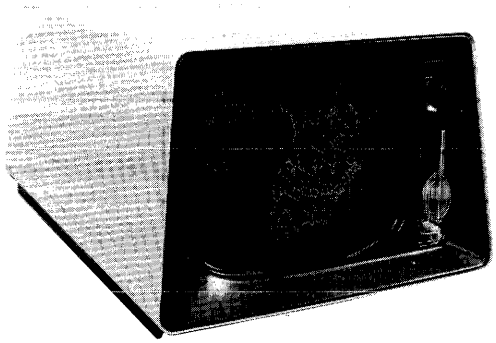


Paper Tape Reader/Punch

Figure 1–5 Peripheral Devices



Line Printer



VT-11 Display

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

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 de-

vices. 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; these programs are strictly business-related or laboratory-related.

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.

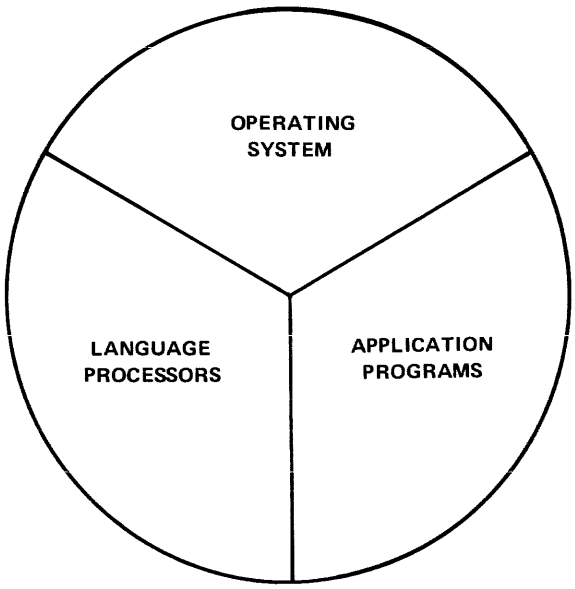


Figure 1-6 RT-11 System Software

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 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.

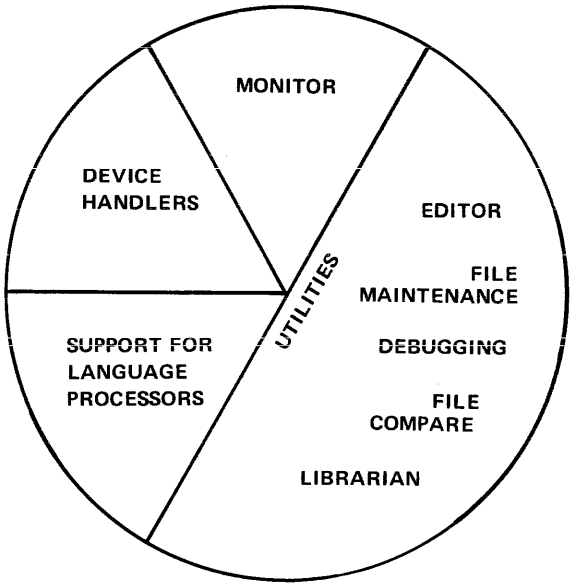


Figure 1-7 RT-11 Operating System

The monitor (executive) program is the interface 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 programs, and locate user-programming errors. Some specific 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.[1]

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 whole 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. High-level languages supported by the RT-11 operating system, in addition to FORTRAN and BASIC, include APL and 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.

Application 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 users) provides a large selection of mathematical and statistical routines commonly required in scientific programming. And a graphics support

[1]Language selection is discussed in Chapter 8 of this manual.

package for BASIC and FORTRAN 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 and final component of a computer system is its documentation. This includes manuals that tell how you use the software and hardware of the computer system, plus any source listings of actual 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 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 that involve the devices.

Software Manuals

Software manuals[1] 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); computer manuals (intended to be used at the computer); and desk manuals (intended to be used at your desk for reference purposes).

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.

Computer 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 computer manual.

Desk 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 manual.

---

[1] All RT-11-related software manuals are listed in the *RT-11 Documentation Directory*. Many of these manuals are provided with your system; others can be ordered from the DIGITAL Software Distribution Center.

Source Listings

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 micro-fiche film from the DIGITAL Software Distribution Center.

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.

REFERENCES

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.

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 Computer Family, Products and Services. Maynard, Mass.: Digital Equipment Corporation, 1977.

An overview of the available 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.

PDP-11 Peripherals Handbook. Maynard, Mass.: Digital Equipment Corporation, 1978.

A technical summary of the available 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, 1978.

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. Maynard, Mass.: Digital Equipment Corporation, 1978.

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

RT-11 Documentation Directory (AA-5285E-TC). Maynard, Mass.: Digital Equipment Corporation, 1980.

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

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.

COMPUTER MEMORY

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

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).

HARDWARE CONFIGURATION

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 various hardware manuals provided with your system and to any special instructions left by the DIGITAL representative who installed your system.

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 Installation and System Generation Guide*.

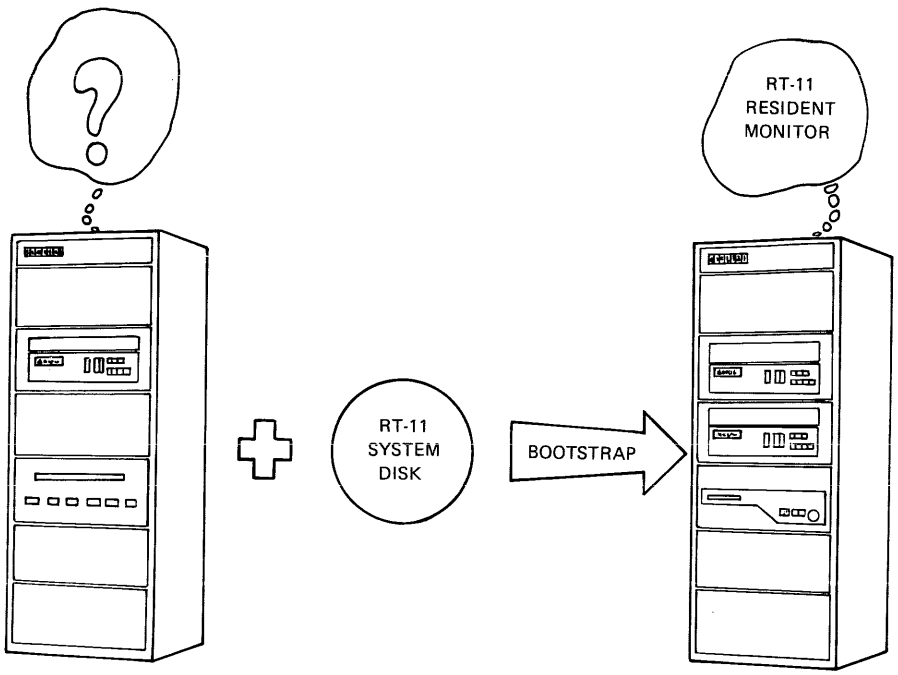


Figure 2-1 Bootstrap/Computer Relationship

NOTE

If your system device is one of the small devices (diskette or DECtape II), 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 Installation and System Generation Guide* contains the information 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 disk device containing the RT-11 operating system (called the system volume); a system volume may have been created specifically for your use with this manual
- The volume containing the FORTRAN and/or BASIC language processors if these languages are not stored on the system volume (available only to FORTRAN and BASIC users)

- A volume for program storage (for example, magtape or another disk); 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 Installation and System Generation Guide*

NOTE

Hardware configuration information, along with instructions for starting (bootstrapping) your RT-11 system, should have been provided by the DIGITAL representative who initiallly installed your system. This information should appear in the *RT-11 Installation and System Generation Guide* and 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.

Terminal

1. What kind of terminal device are you using (for example, LA36 DECwriter II, VT52 video terminal, etc.)?

Computer

2. Does your computer operator's console have pushbuttons or switches?

3. How much memory does your computer have?

System Volume

4. What kind of system volume are you using (for example, RK06 disk, RX01 diskette, etc.)?

5. What is the two-letter code for this volume (typical codes are given in Table 2-1; respond with the code for your own volume)?

Table 2-1 Representative System Volumes

Volume	Code
RX01 diskette	DX
RX02 diskette	DY
RK05 disk	RK*
RK06/07	DM
RP02/03 disk	DP
RL01 disk	DL
RF11 disk	RF
RJS03/4 disk	DS
TC11 DECtape	DT
TU58 DECtape II	DD

*Use DK to bootstrap from an RK05 disk. See steps 7 and 10 in the Bootstrap Procedures section of this chapter.

Storage Volume

6. What volume are you using for program storage (for example, TM11 magtape, RK05 disk)?

7. In which device unit will you use this volume (choose any available device unit — for example, 0, 1, etc.)?

Optional Devices and Supported Languages

8. 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)?

9. 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, making the RT-11 computer system available for you to use.

1. Turn the terminal to an on-line condition.

2. Make sure 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 its corresponding device unit 0. Make sure that the system volume is write-protected (for all except RX01 diskette, which is always write-enabled).

4. Place the storage volume in the device unit noted in question 7 in the Hardware Configuration section. Make sure that this volume is write-enabled.

5. Check the operator's console on your computer (refer to question 2 in the Hardware Configuration section). If your console has pushbuttons, continue. 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 section in Appendix A entitled Using a Pushbutton Console to Bootstrap; otherwise, continue to step 7.

7. Your terminal printer or screen should show several numbers followed by:

   ```
   $
   ```

   Type on the terminal keyboard the two-letter code that represents your system volume (from question 5 in the Hardware Configuration section) followed by a carriage return (the RETURN key), represented throughout the text by the characters (RET)[1]. Be sure to use the SHIFT key so that you type upper-case characters. For example, for RX01 diskette, type:

   ```
   DX (RET)
   ```

   Continue to step 11.

[1]The RK05 disk is an exception. Hardware bootstraps use DK, not RK, for RK05.

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 section in Appendix A entitled Using a Switch Register Console to Bootstrap.

9. Set the three-way dial to ~~DC~~ AUX 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 it is:

      $

      type on the terminal keyboard the two-letter code that represents your system volume (from question 5 in the Hardware Configuration section) followed by a carriage return (the RETURN key), represented throughout the text by the symbol (RET). Be sure to use the SHIFT key so that you type uppercase characters. For example, for RX01 diskette, type:

      DY
      DX (RET)

      Continue to step 11.

    - Any other response indicates that you must type the bootstrap on the terminal keyboard. Read the section in Appendix A entitled Typing the Bootstrap on the Terminal Keyboard.

11. If your system has been correctly bootstrapped, a message prints on the console terminal. Check this message; it should read:

    RT-11SJ V04.xx (the xx's have developmental significance only and can be ignored)

    If this version number (with the exception of the xx's) does not appear, read the section in Appendix B entitled Suggestions for Bootstrapping the System.

The proper response indicates that the monitor component of the RT-11 operating system is active. Set the system volume to a write-enabled condition (for all except RX01 diskette, which is always write-enabled).

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

REFERENCES

DECscope Users' Manual[1] (EK-VT5X-OP-001). Maynard, Mass.: Digital Equipment Corporation, 1975.

A manual for the owners and operators of the DECscope (VT50) family of video terminals 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.

PDP-11 Processor Handbook, Maynard, Mass.: Digital Equipment Corporation, 1978.

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.

*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 this device.

RT-11 Installation and System Generation Guide (AA-H376A-TC) and *RT-11 System Release Notes* (AA-5286C-TC). Maynard, Mass.: Digital Equipment Corporation, 1980.

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

[1]Used as an example; consult hardware user or maintenance manuals specific to your system.

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.

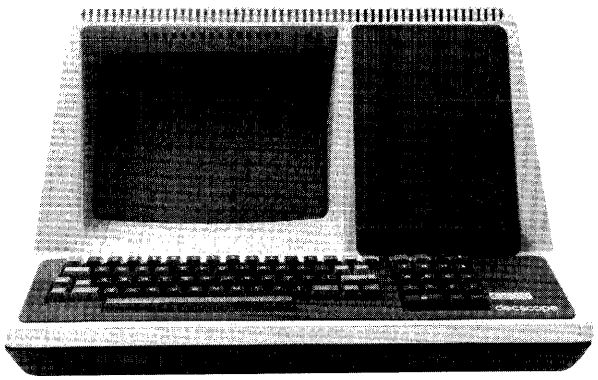
USING THE CONSOLE TERMINAL TO EXCHANGE INFORMATION

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:

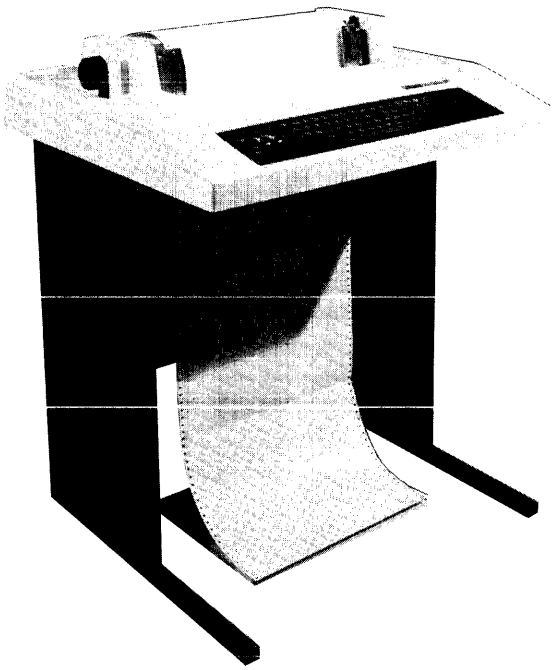
- Type the commands that control system operation
- Receive messages and responses from the system

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 LA36 and the ~~VT52.~~

These two terminals differ in their output mechanism. While the LA36 terminal has only a paper printer, the VT52 has a video screen. The screen and the paper printer 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 the same and are shown in Figure 3-2. Also shown in this figure is an LA30 (VT05) keyboard so that you can note some of the differences found in the keyboards of older terminals.

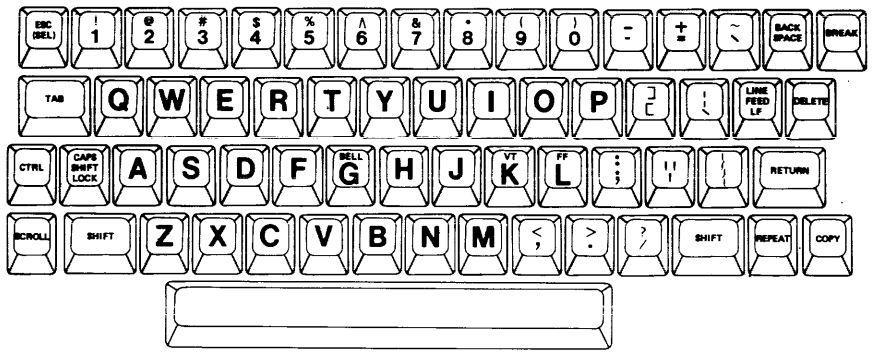


VT52

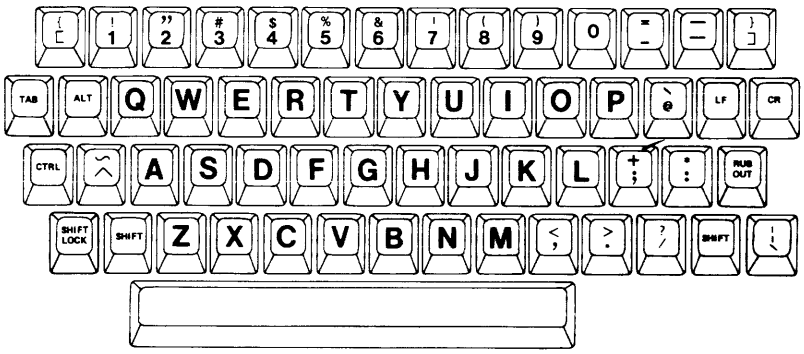


LA36

Figure 3-1 LA36/VT52 Terminals



VT05/LA30 Keyboard



VT52/LA36 Keyboard

Figure 3-2 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 upper- and lower-case characters.[1] 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 (LA36/VT52, VT100 terminals) or the RUBOUT key (LA30/VT05 terminals). These keys perform the same function: they are 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.

---

[1]With the exception of system messages and one other exception explained in Chapter 5, the RT-11 computer system uses upper-case 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 (LA36/VT52, VT100 terminals) or CR (LA30/VT05 terminals) 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 (SEL) key and LINE FEED key (LA36/VT52, VT100 terminals) and ALT and LF keys (LA 30/VT05 terminals). These are special command terminators that are described later in Chapters 5 and 14.

An important key is the CTRL key. The CTRL key is always used in conjunction with another character key to perform one of several specific 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 specifically mentioned are not used by the RT-11 computer system and can be ignored.

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

USING MASS STORAGE VOLUMES TO STORE INFORMATION

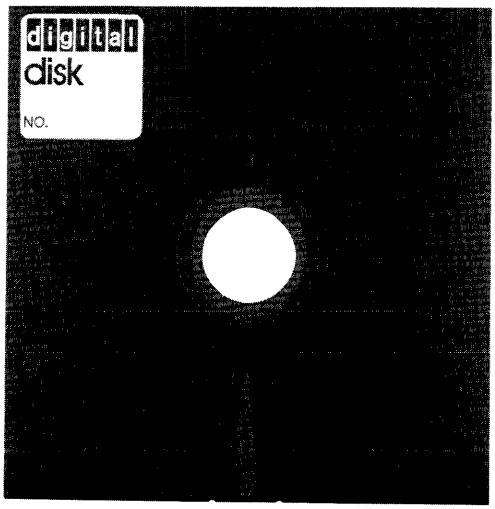
Mass storage volumes provide an area (apart from computer memory) to keep information for later use. The information may be user application programs, data needed by a program, the results of a program run, textual information, batch-type programs, and so on. As an example, the RT-11 operating system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

Before you can access the information stored on any storage volume, however, you must first insert the volume (the medium) into its corresponding device unit (drive), which is the hardware device connected to the computer. Once a volume has been inserted into a device unit, the unit's symbol also identifies the volume. There may be more than one device unit for any given volume, in which case 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 this device unit as long as you are using the system. Other storage

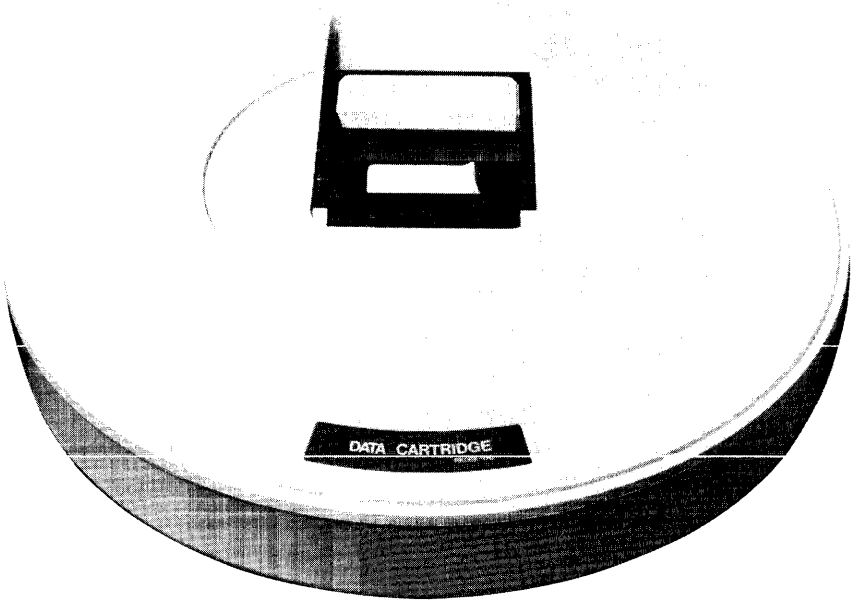
volumes can be inserted in any available (corresponding) device units. Figure 3-3 illustrates several mass storage volumes.

Table 3-1 Keyboard Characters

Key	Function
ALT ALTMODE	See ESC
BACK SPACE	Ignored during normal system use
BREAK	Ignored during normal system use
CR	See RETURN
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
LF LINE FEED	Command terminator; terminates certain system commands; transmits the command to the computer and performs a carriage return
NEW LINE	See LF
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
RUBOUT	See DELETE
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

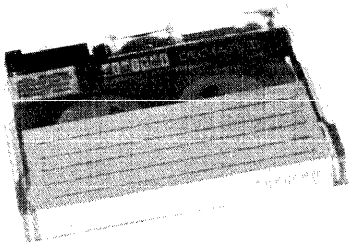


Diskette

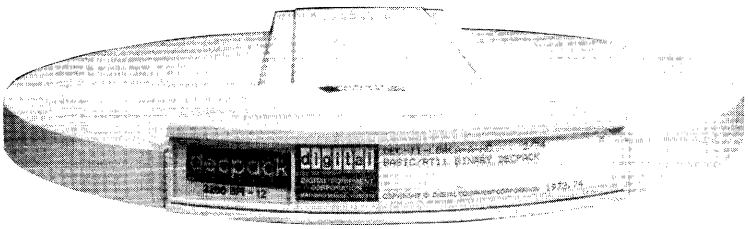


RK06

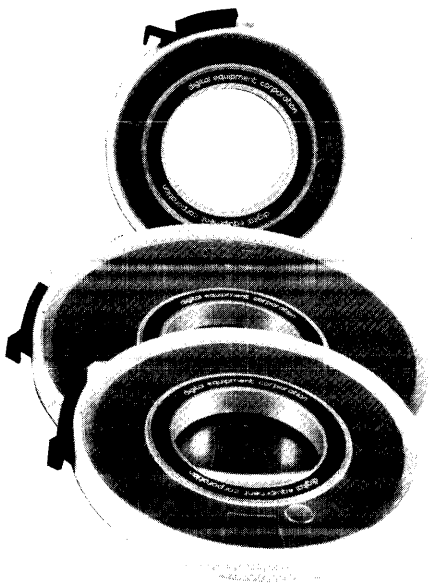
Figure 3–3 Mass Storage Volumes



DECtape II



RK05



Magtape

Figure 3-3 Mass Storage Volumes (Cont.)

Mass storage volumes are capable of holding large amounts of information. However, most volumes 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 all disks (shown earlier in Figure 1-4), magtapes and cassettes are also mass storage volumes.

File Storage

You store information on a mass storage volume in the form of files. Each file is simply 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 physical area of the volume.

Every file on a mass storage volume has a unique name that is composed of a file name and file type. The file name and file type serve to 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 any given 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 residing on that volume and erase 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 Protection

Occasionally, after many files are added to a storage volume, the volume runs out of room for new information. The 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 is called a backup file and insures you against the loss or damage of your original file (or its respective storage volume).

In addition, 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 manually 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.

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 start performing 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

DECscope Users' Manual[1] (EK-VT5X-OP-001) Maynard, Mass.: Digital Equipment Corporation, 1975.

A manual for the owners and operators of VT-50 and VT-52 video terminals and for those who will be programming the computer 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.

*LA36/LA35 DECwriter II User's Manual** (EK-LA3635-OP-001) Maynard, Mass.: Digital Equipment Corporation, 1975.

A hardware manual for the owners and operators of the LA36/LA35 DECwriter II and for those who will be programming the computer to interact with these devices.

RT-11 System Message Manual (AA-5284C-TC) Maynard, Mass.: Digital Equipment Corporation, 1980.

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

[1]Used as an example; consult hardware user or maintenance manuals specific to your system.

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, part of the monitor is called the resident monitor (RMON) which provides console terminal service and centrally required program code to provide 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.

ENTERING COMMAND INFORMATION

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. There are two general formats that you can use to type a command; one is a long format and the other 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. This format allows you to enter all required information on a single command line; prompts are issued only if you neglect to supply necessary information. Both formats are demonstrated throughout this manual.

You terminate all monitor commands by typing 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.

The monitor prints a period at the left margin of the terminal printer or screen whenever it is waiting for you to type a command. The period is your cue that the system is in monitor command mode and ready to accept a monitor command. Check the

output on your terminal printer or screen. You should see at the left margin:

```
RT-11SJ     V04.xx
.
```

RT–11SJ identifies the RT–11 monitor called the single-job (SJ) monitor. Following this is the version (and update) number of the system in use, in this case, Version 4. The period on the next line indicates that the system is in 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][1]	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.

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

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

Long Command Format (system prompts for specific information)

.COMMAND[/option] RET
INPUT PROMPT? INPUT[/option] RET
OUTPUT PROMPT? OUTPUT[/option] RET

Short Command Format (no prompts)

.COMMAND[/option] INPUT[/option] OUTPUT[/option] RET

Notice that you use a slash (/) character to separate an option from the portion of the command that it qualifies, and a carriage return RET to terminate each individual command line. When you have supplied all the necessary information, the carriage return signals the monitor to execute the command. You may use whichever format you wish. Both command formats are demonstrated throughout the manual.

Control Commands

In addition to monitor commands, there are several special function commands, called control commands, that you type by first pressing the CTRL key on the terminal keyboard, and then (while holding it down) typing the appropriate letter key of the command. These commands require no terminator; the system performs the function as soon as you type the command.

Control commands are special function commands 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.

Recreating the Examples

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 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
(RET)	carriage return key
(LF)	line feed key
(SP)	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.
(TAB)	TAB key (once for each time the symbol is shown)
(DEL)	DELETE (RUBOUT) key (once for each time the symbol is shown)
(ESC)	ESCAPE (ALTMODE) key (once for each time the symbol is shown)
(CTRL/x)	CTRL key (hold down CTRL key while typing the letter character [x])

CORRECTING TYPING MISTAKES

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

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.

CTRL/U

The second way to correct a typing error is to use a special control command, CTRL/U. 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 the CTRL key and then type the U key while holding the CTRL key down; 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 effectively 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 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 via 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.

INITIAL MONITOR COMMAND OPERATIONS

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 and you decide that you want to use it, you should also enable (turn on) the graphics display screen.

Using VT11 Display Hardware

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[1], which is illustrated in Figure 4-1, you can use the graphics screen in place of the terminal printer or screen if you wish.

[1]Video terminal screens are not considered graphics display hardware.

NOTE

Check question 5 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 in this chapter, Entering the Date and Time-of-Day.

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.

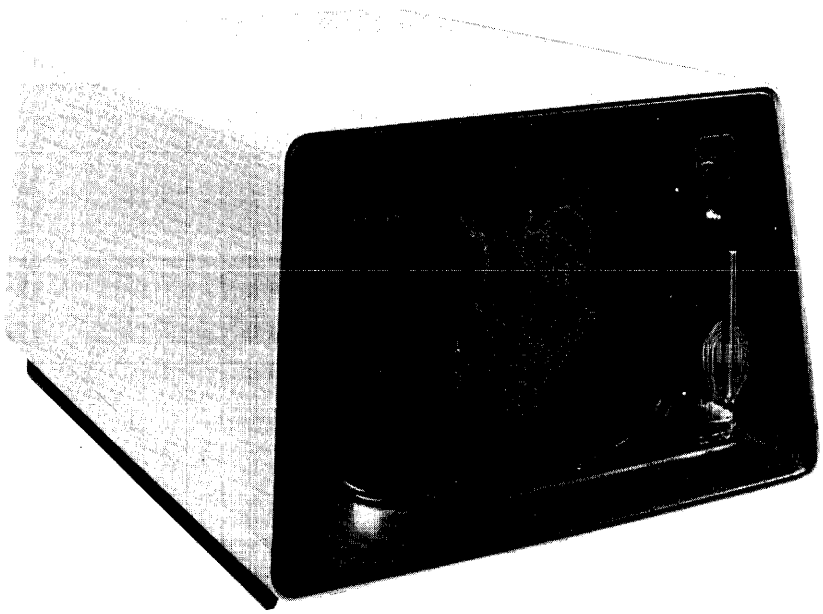


Figure 4-1 VT11 Display Hardware

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[1] on the terminal printer or screen informing you that the command is illegal for your system configuration:

```
?KMON-F-Illegal command
```

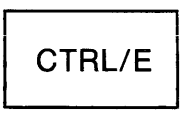
[1]The meanings of all system messages are listed in the *RT-11 System Message Manual.*

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.)

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.

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

Entering the current date and time-of-day are record-keepng system operations; they help you later identify when other 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.

DATE

Enter the date by typing the monitor DATE command as follows (there is only one format):

Long and Short Command Format

```
.DATE 8-JAN-80 (RET)
```

This sets the date to January 8, 1980. Since this date is not current, reenter the correct date using the same command format:

```
.DATE dd-mmm-yy (RET)
```

Typing the new date overrides the previous date.

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 (RET)
```

If your system does not have a clock, the monitor prints a message on the terminal informing 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 (RET)
```

Typing the new time overrides the previous time.

To check the time and 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-80

.TIME (RET)
15:06:19
```

The system responds by printing the date or the time, based on the information you previously entered.

Setting the time is temporary. If you want it 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.

Assigning Logical Names to Devices

Each hardware device in the RT-11 system is identified by a two-character code name. These names, 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.

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

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 codes 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 DX: or DX0:; diskette unit 1 is DX1:; RK: disk unit 2 is RK2:; and so on. Note that, according to Table 4–3, you can use the device codes 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.

Table 4–2 Physical Device Names

Code	Device
CR:	Card Reader
CTn:	Cassette
DDn:	DECtape II
DLn:	RL01 Disk
DMn:	RK06/07 Disk
DPn:	RP02/03 Disk
DSn:	RJS03/4 Disk
DTn:	DECtape
DXn:	RX01 Diskette
DYn:	RX02 Diskette
LP:	Line Printer
MMn:	TJU16 Magtape
MSn:	TS11 Magtape
MTn:	TM11 Magtape
PC:	Paper Tape Punch/Reader
RF:	RF11 Disk
RKn:	RK05/RK11 Disk
TT:	Console Terminal Keyboard/Printer

Table 4–3 Special Logical Device Names

Code	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).

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.

There are many reasons why you might want to change a device name temporarily and assign it a logical name. You may, for example, have a program that is written for a specific device that

is not available on your system. If you assign its name to a device that is available, the program then uses the new device instead.[1]

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 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 6 and 7 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

SHOW

[1]This is called device independence.

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. For example, use the SHOW command now to check the status of the assignment just made:

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
SY:RK   (Resident)
        RK0 = SY , DK
        RK1 = VOL
BA
MT
NL
LS
LP
DD
DM
DX
DT
5 free slots
```

Logical device assignments are temporary. Thus, if you want a particular 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.[1] For example, to list the directory of your system volume, type:

Long and Short Command Format

```
.DIRECTORY RET (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:

CTRL/O

[1]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.

This special control command echoes as ^O and 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:

```
 08-Jan-80
SWAP   .SYS    25  19-Nov-79      RT11SJ.SYS    67  19-Nov-79
RT11FB.SYS    80  19-Nov-79      RT11BL.SYS    64  19-Nov-79
TT     .SYS     2  19-Nov-79      DT     .SYS     3  19-Nov-79
DP     .SYS     3  19-Nov-79      DX     .SYS     3  19-Nov-79
DY     .SYS     4  19-Nov-79      RF     .SYS     3  19-Nov-79
RK     .SYS     3  19-Nov-79      DL     .SYS     4  19-Nov-79
DM     .SYS     5  19-Nov-79      DS     .SYS     3  19-Nov-79
DD     .SYS     5 ^O

 189 Files, 3785 Blocks
 977 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

Table 4-4 File Types

File Type	Meaning
.BAC	BASIC compiled file
.BAK	Editor backup file
.BAS	BASIC source file
.BAT	BATCH source file
.COM	Indirect command file
.CTL	BATCH control file
.DAT	BASIC or FORTRAN data file
.DBL	DIBOL source file
.DIF	SRCCOM output file
.DIR	Directory listing file
.FOR	FORTRAN source file
.LOG	Batch log file
.LST	Listing file
.MAC	MACRO source file
.MAP	Linker map file
.OBJ	MACRO, FORTRAN, or DIBOL object output file or library file
.REL	Executable foreground program file
.SAV	Executable background program file
.SML	System MACRO library
.SYS	System files and handlers

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 command) 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 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 or give a protected status to a file by using the RENAME keyboard monitor command with the /PROTECTION or /NOPROTECTION option (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-80
SWAP   .SYS     RT11SJ.SYS   RT11FB.SYS     RT11BL.SYS     TT     .SYS
DT     .SYS     DP     .SYS  DX     .SYS     DY     .SYS     RF     .SYS
RK     .SYS     DL     .SYS  DM     .SYS     DS     .SYS     DD     .SYS
CTRL/C CTRL/C
```

DIRECTORY /PRINTER

Volume directories can be printed on a line printer if one is available on your system. Check question 8 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 entire listing may be quite long. When the line printer is done printing, retrieve the listing.

Initializing the Storage Volume

Initializing a storage volume completely 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 the effect of an initialize operation is to remove all file names from the directory. So before you initialize any volume, be sure that there are no files on it 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 ensures that there is room on the volume for new files.

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?* is always printed to give you an opportunity to verify the command. Typing a Y initiates the operation, while N aborts (ignores) the operation and returns control to 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-80

0 Files, 0 Blocks
4762 Free block
```

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 dn:
List the volume directory on the terminal (dn: is the code for the device name; the default storage volume (DK:) is assumed if dn: is not specified).

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

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

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

INITIALIZE dn:
Clear the directory of the indicated volume (dn: is the code for the device name and must be specified).

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.

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).

SUMMARY: SPECIAL CONTROL COMMANDS

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.

REFERENCES

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 Pocket Guide (AV-5287C-TC). Maynard, Mass.: Digital Equipment Corporation, 1980.

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

RT-11 System User's Guide (AA-5279B-TC). Maynard, Mass.: Digital Equipment Corporation, 1980.

A guide to the use of the RT-11 operating system. See Chapters 3 and 4.

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.

THE RT-11 EDITOR

The RT-11 editor is a system utility program called EDIT.SAV, which is stored as part of the RT-11 operating system on your system volume. 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 the appropriate ASCII codes; when you request listings on the terminal or line printer, the system converts the ASCII code back to the appropriate 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 existing 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).

Since the text buffer is a finite 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.

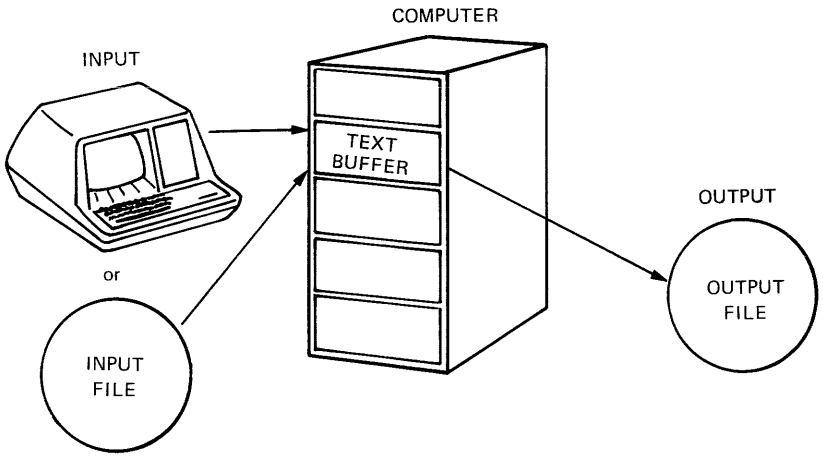


Figure 5-1 Editing with RT-11

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 codes 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 the 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 control is in editing command mode and is your cue to enter an editing command.

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

INSERT

```
*I
```

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 (ALTMODE) 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 ENDOWED 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 currently in the text buffer to the output file, closes the currently open output 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:

EXIT

```
*EX (ESC)(ESC)
```

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 an existing file, you again use the EDIT command to activate the editor. Next indicate in the command line the two-character device code 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 ESC ESC
*
```

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
*
```

LIST

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.

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 TRUTS 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.
*
```

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

```
.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 TRUTS 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[1]:

[1] 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.

```
*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
*
```

VERIFY

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,
*                     ↑
```

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:

```
*7J ESC L ESC ESC
E SELF-EVIDENT,
*
```

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:

```
*7OJ(ESC)ISTART A(RET)
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.[1] 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):

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

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.

ADVANCE

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:

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

KILL

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.

[1]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.

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:

```
*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
*  ↑
```

GET

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

```
*GOR(ESC)I(SP)WITH(SP)CERTAIN(RET)
INALIENABLE(SP)RIGHTS(ESC)-A(ESC)2L(ESC)(ESC)
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 mis-

spelling. 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:

```
                ESC
*GAND(RET)I(SP)THE(SP)PURSUIT(SP)OF(ESC)(ESC)
*GPL(ESC)-2D(ESC)IPI(ESC)V(ESC)(ESC)
THESE ARE LIFE, LIBERTY AND THE PURSUIT OF HAPPINESS.
*
```

CTRL/L

Large text files of 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.(ESC)I(RET)
(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:

```
*GAMONG(ESC)I(SP)THESE(SP)ARE(ESC)A(ESC)I0D(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.

*
```

NEXT

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.

```
*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. .BAK identifies it as 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-80
DECIND.BAK      1  19-Nov-79   DECIND.USA  1  08-Jan-80
 2 Files, 2 Blocks
 790 Free blocks
```

The * 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 UPPER- AND LOWER-CASE CHARACTERS

Edit Lower

Later model terminals (for example, LA36 DECwriters and VT52 DECSCOPE terminals) have the capability to print in upper and lower case. Certain line printers also have this capability. You can use the upper-/lower-case capability of these devices if you type the EL (Edit Lower) editing command before entering the text you want to insert in lower case. 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 upper case, is disabled. In addition, the characters are echoed on the terminal printer or screen as upper- and lower-case characters.

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

Long and Short Command Format

```
.EDIT DECIND.USA<RET>
*EL<ESC><ESC>
*
```

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

```
*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.
*
```

Edit Upper

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

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

```
*eu<ESC><ESC>
*
```

Upper-case 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 upper-/lower-case capability.

Close the file DECIND.USA by typing:

```
*EX ESC ESC
*
```

SUMMARY: EDITING COMMANDS

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.

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 may 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 may 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 may 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 may be used)

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

Upper-/Lower-Case Commands (pointer is not affected)

EL
Edit Lower; accept characters typed at the keyboard as upper/lower case.

EU
Edit Upper; revert to upper-case editing (after EL).

USING A GRAPHICS DISPLAY TERMINAL DURING EDITING

If your system configuration included VT11 display hardware, there are several advantages to your using it during editing.[1] 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.

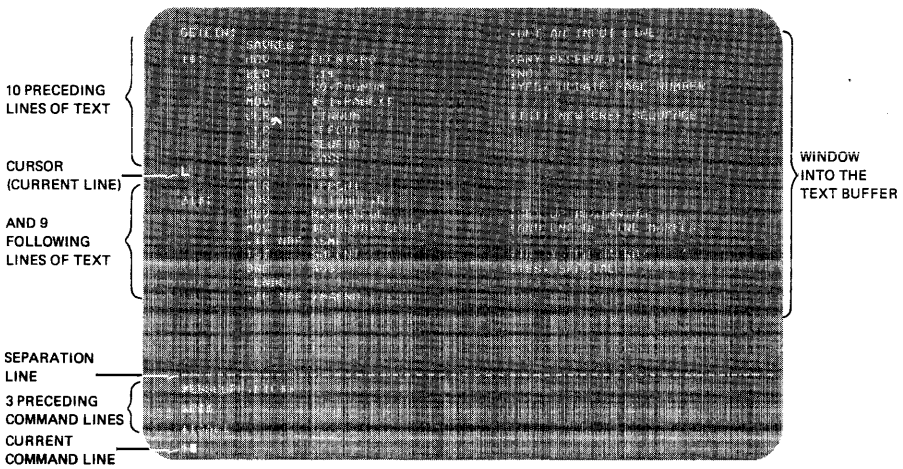


Figure 5–2 Text Window Format

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)
```

[1] If your system does not have VT11 display hardware, skip to the next section, Creating the Demonstration Programs.

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. However, 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.TXT↵
*
```

ESCAPE ESCAPE

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.

```
*ESC ESC
!
```

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

Character Insertion

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.

The next several paragraphs demonstrate the use of the immediate mode commands on a selected portion of text. Remember that all characters you type that are not immediate mode commands are treated as input. Commands do not echo on the graphics screen, so all you ever see is the current text file. Type the following:

```
TO BE, OR NOT TO BE — THAT IS THE QUESTION:RET
WHETHER 'TIS NOBLER IN THE MIND AND HEART TO SUFFERRET
THE SLINGS OF OUTRAGEOUS FORTUNERET
OR TO TAKE ARMS AGAINST A SEA OF TROUBLES,RET
AND BY OPPOSING END THEM?RET
```

CTRL/G

As you can see on the graphics screen, the cursor (pointer) is positioned at the beginning of a new line. CTRL/G, equivalent to -A in standard editing, moves the cursor to the beginning of the previous line; the cursor is repositioned immediately. Type:

```
CTRL/G
CTRL/G
CTRL/G
```

The cursor has moved backward three lines, one line for each CTRL/G command, and is positioned before the line:

```
THE SLINGS OF OUTRAGEOUS FORTUNE,
```

CTRL/V

CTRL/V, equivalent to -J, moves the cursor back one character. Move the cursor back over the carriage return and line feed at the end of the previous line by typing the CTRL/V command eleven times (remember, the carriage return and line feed count as two characters):

```
CTRL/V                                  (eleven [11] times)
```

```
WHETHER 'TIS NOBLER IN THE MIND AND HEART TO SUFFER
```

DELETE

This positions the cursor before the word TO. The command DELETE, equivalent to -D, deletes the character immediately preceding the cursor. Type the DELETE key ten times:

```
DEL                                     (ten [10] times)
```

```
WHETHER 'TIS NOBLER IN THE MIND TO SUFFER
```

CTRL/N

CTRL/N, equivalent to A, advances the cursor to the beginning of the next line:

CTRL/N

THE SLINGS OF OUTRAGEOUS FORTUNE,

CTRL/D

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

CTRL/D (ten [10] times)

THE SLINGS OF OUTRAGEOUS FORTUNE,

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

SP AND SP ARROWS

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 OUTRAGEOUS 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.

ESCAPE

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

CTRL/C ESCAPE ESCAPE

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 ESC ESC

CREATING THE DEMONSTRATION PROGRAMS

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.

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 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), 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 ESC ESC before closing the file.

Create the FORTRAN file first. Call it GRAPH.FOR and use the system volume for storage. Then create the MACRO 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 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

        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/
        SCAL(ZMIN,ZMAX,MAXZ,K)=ZMIN+FLOAT(K-1)*(ZMAX- ZMIN)/FLOAT(MAXZ-1)
        CALL SCOPY('- 1 2 3 4 5 6 7 8 9 +',STAB)
        MAXFLEN(STAB)
        DO 20 IX=1,MAXX
        IIX=IX
        X=SCAL(XMIN,XMAX,MAXX,IIX)
        CALL REPEAT('*',STRING,MAXY)
        IF(IIX.EQ.1 .OR. IX.EQ.MAXX) GOTO 20
        DO 10 IY=2,MAXY-1
        IIY=IY
```

```
        Y=SCAL(YMIN,YMAX,MAXY,IIY)
        IFUN=2+INT(FLOAT(MAXF-3)*(FUN(X,Y)-FMIN)/(FMAX-FMIN))
10      STRING(IIY)=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 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 RECIPROCALS 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
        BCC     FIFTH           ;(REALLY DIVIDE BY 10)
        ADD     #10+'0,R0       ;MAKE DIGIT ASC II
        .TTYON                  ;OUTPUT THE DIGIT
        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.

REFERENCE

RT-11 System User's Guide (AA-5279B-TC). Maynard, Mass.: Digital Equipment Corporation, 1980.

A guide to the use of the RT-11 operating system. See Chapter 5.

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 (SUM.MAC and GRAPH.FOR) 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.

PERFORMING A COMPARISON

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.

DIFFERENCES

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

Long Command Format

```
.DIFFERENCES/MATCH:1 (RET)
File     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 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,R0               ;GO THRU ALL PLACES,
                                                 ;DIVIDING
B ****
C 2)1            BNE       2ND                 ;BRANCH IF NOT
D 2)             MOV       #N,R0               ;GO THRU ALL PLACES,
                                                 ;DIVIDING
  **********
C 1)1            ADD       #10+'0,R0           ;MAKE DIGIT ASCII
D 1)             .TTYON                        ;OUTPUT THE DIGIT
B ****
C 2)1            ADD       #10+'0,R0           ;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
```

[1]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 is 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 B).

DIFFERENCES/
MATCH:n

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 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 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.

A message is printed following the comparison. *Files are different* is printed if differences exist; *No differences encountered* is printed if the files are exactly 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 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 (RET)

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 in Chapter 5 entitled Editing a Text File. 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 only finds differences 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
File 1? DEMOX1.MAC
File 2? SUM.MAC
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 seem to resolve them by reediting, 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.

SUMMARY: COMPARISON COMMAND

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.

REFERENCE

RT-11 System User's Guide (AA-5279B-TC). Maynard, Mass.: Digital Equipment Corporation, 1980.

A guide to the use of the RT-11 operating system. See Chapters 4 and 15.

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 have a 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.

FILE DIRECTORY OPERATIONS

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.

Long and Short Command Formats

(Line printer)

```
.DIRECTORY/PRINTER RET
```

(Terminal)

```
.DIRECTORY RET
```

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-80
DECIND.BAK          1   8-JAN-80
GRAPH .FOR          2   8-JAN-80
GRAPH .BAK          2   8-JAN-80
SUM   .MAC          3   8-JAN-80
SUM   .BAK          3   8-JAN-80
```

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 Formats

(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 a shorthand method provided by the RT-11 operating system called the wildcard construction. This 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. However, it is valid only for the file maintenance commands listed in this chapter; the wildcard construction is not valid for any other commands.

FILE COPYING OPERATIONS

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).

COPY

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; it 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, the original version is not physically moved to the new volume, but a copy of it is made there.

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

Long Command Format

```
.COPY (RET)
From? GRAPH.FOR (RET)          (System volume is assumed for input.)
To  ? VOL:GRAPH.TWO (RET)
```

Short Command Format

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

The system makes an exact 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.

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:[1]

Long and Short Command Formats

```
.DIRECTORY VOL: (RET)
 08-Jan-80
GRAPH .TWO      2  08-Jan-80       SUM    .MAC      3  08-Jan-80
DEMOX1.MAC      3  19-Nov-79       DEMOF1.FOR      2  19-Nov-79
 4 Files, 10 Blocks
 4752 Free blocks
```

[1]If you are using magtape or cassette as your storage volume, read the section in Appendix B entitled Directory vs Nondirectory-Structured Volumes.

FILE RENAMING OPERATIONS

RENAME

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 type, .FOR. Use the monitor RENAME command to rename the file already on the storage volume.[1]

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 Formats

```
.DIRECTORY VOL: (RET)
 08-Jan-80
GRAPH .FOR      2  08-Jan-80       SUM    .MAC      3  08-Jan-80
EXAMP .MAC      3  19-Nov-79       EXAMP  .FOR      2  19-Nov-79
 4 Files, 10 Blocks
 4752 Free blocks
```

[1]Magtape and cassette users cannot use the RENAME command and should read Appendix B, Alternate RENAME Operation for Magtape and Cassette Users.

FILE DELETION OPERATIONS

Once copies of your important files are stored on a storage volume, you can delete (erase) from the system (or any other) volume those files that you no longer need. The file deletion operation deletes the entry from the volume directory. Thus 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.

Now that you have copies of your important files, you can delete several file names 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, which therefore should not be erased 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. The DELETE command defaults to requesting confirmation from the user by printing each file name on the terminal before it deletes it. This gives you the opportunity to confirm each file before deletion. If you type a Y response, the system deletes the file name, while an N response instructs the system to ignore that file name and go on to the next. You can specify as many as six input files for deletion. Notice how you use the wildcard construction in one of the input files to delete all files with a .BAK file type.

Long Command Format

```
.DELETE (RET)
Files? DECIND.USA,*.BAK,GRAPH.FOR,SUM.MAC (RET)
 Files deleted:
DK:DECIND.USA ? Y (RET)
DK:GRAPH.BAK  ? Y (RET)
DK:DEMOF1.BAK ? Y (RET)
DK:DECIND.BAK ? Y (RET)
DK:GRAPH.FOR  ? Y (RET)
DK:SUM.MAC    ? Y (RET)
```

Short Command Format

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

```
 Files deleted:
DK:DECIND.USA ? Y (RET)
DK:GRAPH.BAK  ? Y (RET)
DK:DEMOF1.BAK ? Y (RET)
DK:DECIND.BAK ? Y (RET)
DK:GRAPH.FOR  ? Y (RET)
DK:SUM.MAC    ? Y (RET)
```

FILE LISTING OPERATIONS

You sometimes need to obtain 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 obtain listings of the files you created. You can also obtain listings of files using monitor commands. One command lists a file on the console terminal; another lists a file on the line printer.[1] 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.

Long Command Format

PRINT

(Line Printer)

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

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

(Terminal)

TYPE

```
.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)
```

These file maintenance operations are the kinds of operations that you should perform 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.

[1] If a line printer is available on your system, you should always use it for listings. Line printer listings are neater and print faster than terminal listings.

SUMMARY: FILE MAINTENANCE COMMANDS

COPY
Copy the specified file from one volume to another.

DELETE
Delete the specified file(s) from the volume's directory. Confirmation required before deleting the file.

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.

RENAME
Give a new name to the specified file.

TYPE
List the contents of the specified file on the terminal.

REFERENCE

RT-11 System User's Guide (AA-5279B-TC). Maynard, Mass.: Digital Equipment Corporation, 1980.

A guide to the use of the RT-11 operating system. See Chapter 4.

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.

HIGH-LEVEL VS MACHINE-LEVEL LANGUAGES

Hundreds of programming languages have been developed for computer systems. Some languages can be used only for specific applications or in conjunction 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 and FORTRAN IV) and one RT-11 system-specific machine-level programming language (MACRO-11).

High-level languages, like BASIC and FORTRAN, 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) handle 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 the appropriate 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.

On the other hand, machine-level languages, 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 the job done.

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

Table 8–1 Language Comparisons

High-Level	Machine-Level
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

In general, beginning programmers, students, commercial applications programmers, and the casual computer user tend to prefer high-level languages because they are less difficult to learn and use and produce fast results. System programmers, on the other hand, may prefer machine-level languages. The programs they write (those that make up an operating system, for example) 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 they feel 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	
APL	

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, 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.

If your application mainly requires the use of complicated mathematical operations or mixed data types, you may prefer to select the programming language APL. This language uses a concise and powerful shorthand notation to perform arithmetic and logical operations on vectors, matrices, and arrays.

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 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 is probably the best language for you to start with: FORTRAN 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 9 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

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 Computer Family—Software and Services. Maynard, Mass.: Digital Equipment Corporation, 1977.

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

PDP-11 Software Handbook. Maynard, Mass.: Digital Equipment Corporation, 1978.

A general overview and introduction to available PDP-11 software, operating systems, and language processors. See Chapters 1, 2, and 3.

CHAPTER 9
RUNNING A FORTRAN IV PROGRAM

The FORTRAN IV programming language[1] 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.

THE FORTRAN IV PROGRAMMING LANGUAGE

FORTRAN (FORmula TRANslation) is an algebraically-oriented language. You write a FORTRAN 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 comprise the entire executable FORTRAN source program.

When you are satisfied with the logic of your FORTRAN 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 program is represented in Figure 9–1.

Figure 9–1 Evolution of a FORTRAN Program

[1]The PDP-11 FORTRAN IV programming language conforms to the specifications for American National Standard FORTRAN X3.9-1966.

THE FORTRAN IV LANGUAGE PROCESSOR

The FORTRAN IV language processor is a compiler that accepts information in one format (that is, your source program) and translates it into another format (that is, a machine language program). Since you originally use the editor to create a FORTRAN source program in ASCII format, you must next translate it into a machine format that the computer can use. The FORTRAN compiler performs the translation, producing as output a new version of the program in object format, called an object module. You may optionally instruct the FORTRAN compiler to produce a listing of the source program at the same time. Figure 9–2 is a diagram of the compiler's function.

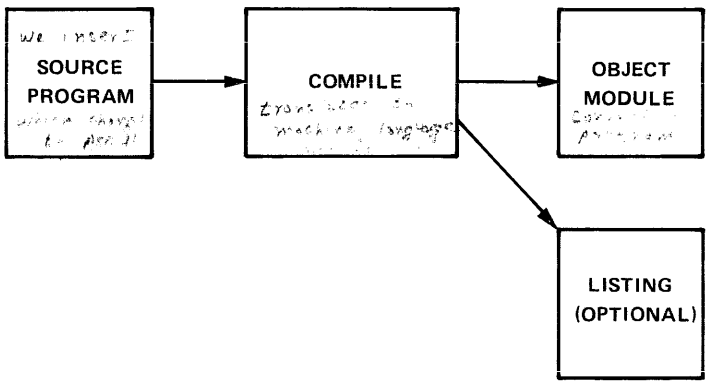


Figure 9–2 Function of a FORTRAN Compiler

USING LIBRARY MODULES

Typical FORTRAN IV programs often require similar operations. For example, most programs 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; perform conversions; and other similar kinds of processes. Thus, 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 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 this and makes the appropriate references to SYSLIB. It translates all the information gathered during processing (your converted language statements and the references to SYSLIB) into numerical data called object code, a machine language 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 program.

COMPILING THE FORTRAN IV PROGRAM

In Chapter 5 you used the RT-11 editor to create a FORTRAN 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.[1] 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 files are on your system volume. However, if 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 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 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
```

Short Command Format

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

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:

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

Long Command Format

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

Short Command Format

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

The FORTRAN 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 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

```
.FORTRAN (RET)
Files? GRAPH/LIST (RET)
```

Short Command Format

```
.FORTRAN GRAPH/LIST (RET)
```

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 a total of 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)

```
.PRINT (RET)
Files? GRAPH.LST (RET)
```

(Terminal)

```
.TYPE (RET)
Files? GRAPH.LST (RET)
```

Short Command Format

(Line printer)

```
.PRINT GRAPH.LST (RET)
```

(Terminal)

```
.TYPE GRAPH.LST (RET)
```

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      V02.1-10   TUE 08-Jan-80 12:14:07         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       LOGICAL*1 STRING(13,3),STAB(100)
0002       DATA XMIN,XMAX,MAXX/-5,5,45/
0003       DATA YMIN,YMAX,MAXY/-5,5,72/
0004       DATA FMIN,FMAX/0.0,1.0/
0005       SCAL (ZMIN,ZMAX,MAXZ,K)=ZMIN+FLOAT(K-1)*(ZMAX-ZMIN)/FLOAT(MAXZ-1)
0006       CALL SCOPY('-1 2 3 4 5 6 7 8 9 +',STAB)
0007       MAXF=LEN(STAB)
0008       DO 20 IX=1,MAXX
0009       IIX=IX
0010       X=SCAL(XMIN,XMAX,MAXX,IIX)
0011       CALL REPEAT('*',STRING,MAXY)
0012       IF(IIX.EQ.1 .OR. IIX.EQ.MAXX) GOTO 20
0014       DO 10 IY=2,MAXY-1
0015       IIY=IY
0016       Y=SCAL(YMIN,YMAX,MAXY,IIY)
0017       IFUN=2<INT(FLOAT(MAXF-3)*(FUN(X,Y)-FMIN)/(FMAX-FMIN))
0018 10    STRING(IIY)=STAB(MIN0(MAXF,MAX0(1,IFUN)))
0019 30    CALL PUTSTR(7,STRING,' ')
0020       CALL EXIT
0021       END
FORTRAN IV
 Diagnostics for Program Unit .MAIN.

In line 0002, Error:   Modes of variable "XMIN" and data item differ
In line 0003, Error:   Modes of variable "YMIN" and data item differ
In line 0008, Error:   Reference to undefined statement label
In line 0013, Error:   Reference to undefined statement label
In line 0018, Error:   Wrong number of subscripts for array "STRING"
FORTRAN IV     Storage Map for Program Unit .MAIN.

Local Variables, .PSECT $DATA, Size = 000340 (  112. words)

Name  Type  Offset  Name  Type  Offset  Name  Type  Offset
FMAX   R*4  000230  FMIN   R*4  000224  IFUN   I*2  000316
IIX    I*2  000262  IIY    I*2  000304  IX     I*2  000264
IY     I*2  000302  K      I*2  000256  MAXF   I*2  000260
MAXX   I*2  000276  MAXY   I*2  000300  MAXZ   I*2  000254
```

```
MAXO    I*2  000322  MINO    I*2  000320  X        R*4   000266
XMAX    R*4  000272  XMIN    R*4  000214  Y        R*4   000306
YMAX    R*4  000312  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  FLOAT  R*4  FUN    R*4  INT    I*2  LEN    I*2
PUTSTR       R*4  REPEAT       R*4  SCAL   R*4  SCOPY  R*4

FORTRAN IV    V02.1-10                                    PAGE 001

0001      FUNCTION FUN(X,Y)
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 comprising 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 method of naming the terminal as the output device. This procedure is repeated until all 45 lines of the graph have been printed.

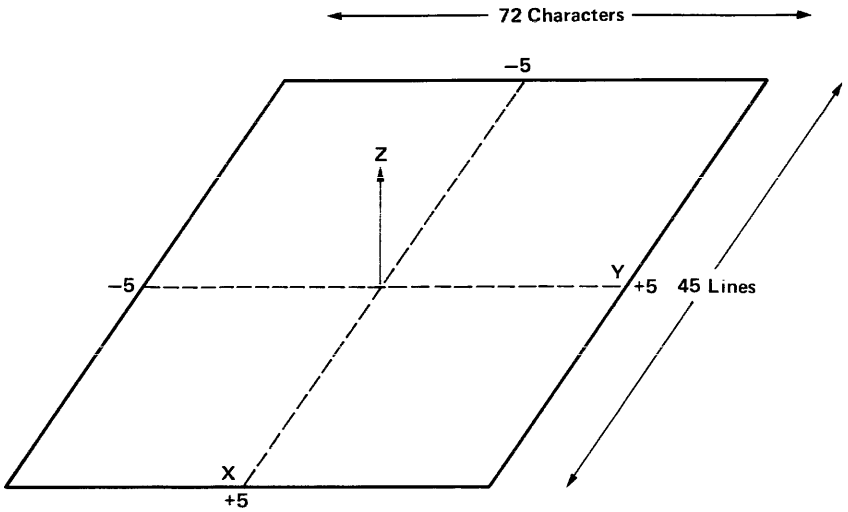


Figure 9-3 Dimensions of FUN(X,Y)

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.[1] Look first at the messages following the main program unit. Errors were discovered in lines 2, 3, 8, 13, and 18.

The messages for lines 2 and 3 indicate that the floating-point variables "XMIN" and "YMIN" are assigned integer values. The DATA statements must be changed. (Note that the same error exists for "XMAX" and "YMAX"; however, the compiler 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 2 and 3), 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 13 to an undefined label. (Line 13 is actually the second portion of line 12, the GO

[1]Refer to the *RT-11 System Message Manual* for greater detail about any system messages printed.

TO statement.) Statement label 20 is referenced in each case, but only labels 10 and 30 are shown in the program. This indicates that either a statement is missing, or that a typing error exists. Examination of the program logic shows a typing error in line 19. Label 30 should actually be 20. Correct line 19 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 1 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 those statements in the source program 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). 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.[1] All required object modules must be joined, or linked together, before the program can work.

Even if your program did 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.

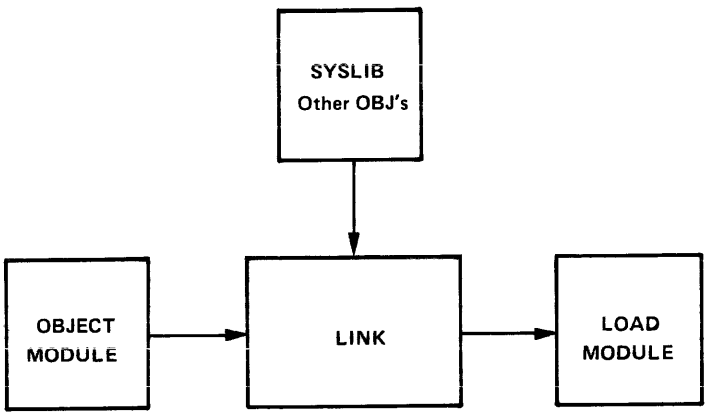


Figure 9–4 The Link Operation

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 always scan the SYSLIB library if it is present on the system volume.

LINK

[1]For more information on linking files and using library files, see Chapters 12 and 13 respectively.

Some RT-11 systems store the linker (LINK.SAV) and the default system library (SYSLIB.OBJ) on a volume apart from the system volume or the FORTRAN/BASIC language volume.[1] You can quickly determine whether the system library is on your system volume by using the DIRECTORY command.

```
.DIRECTORY SY:SYSLIB.OBJ RET
```

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 in Appendix B entitled Using the LINK Volume.

Long Command Format

```
.LINK RET
Files? GRAPH RET
```

Short Command Format

```
.LINK GRAPH RET
```

Any messages printed on the terminal identify error conditions discovered by the system during the link operation (for example, if you fail to specify all the object modules that are needed as input). However, assuming 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.

RUNNING THE FORTRAN IV PROGRAM

If your FORTRAN 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.

[1] 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.

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 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 demonstration program, use the monitor RUN command. You can omit typing the .SAV file type since it is assumed within the RUN command. Type:

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.

COMBINING OPERATIONS

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

EXECUTE

NOTE

In order to use the EXECUTE command, the following files must be present on your system volume:

FORTRA.SAV	LINK.SAV
GRAPH.FOR	SYSLIB.OBJ

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.

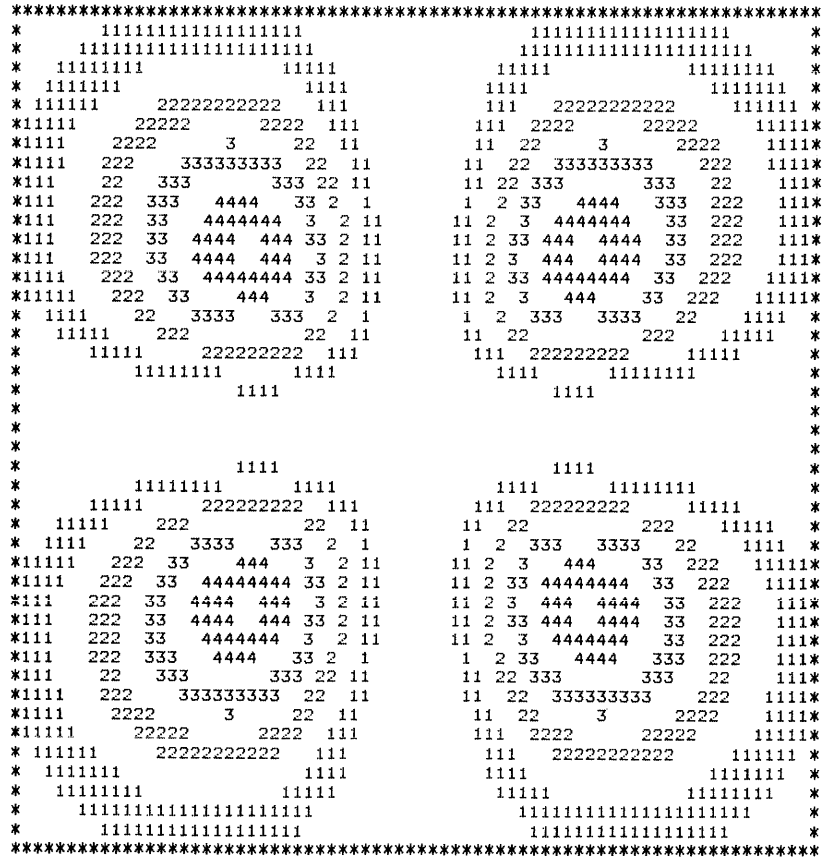


Figure 9–5 The Result of GRAPH.SAV

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 assembler. Another way is to omit the language-name option and explicitly specify the file type for the source files. The EXECUTE command then invokes the language processor that corresponds to that file type. Specifying the file GRAPH.FOR, for example, invokes the FORTRAN 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 it finds neither GRAPH.MAC nor GRAPH.DBL, it invokes the FORTRAN lan-

guage process 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

```
.EXECUTE GRAPH/FORTRAN/LIST (RET)
```

ALTERNATE FUNCTIONS

The following are some alternate functions that you can substitute in your FORTRAN 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 automatically prints on the terminal (or line printer).

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 source program and produce an object module.

FORTRAN/LIST
Compile the FORTRAN 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 demonstration, turn now to the section in Appendix B entitled FORTRAN/LINK File Maintenance.

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

Long and Short Command Format

```
.DIRECTORY GRAPH.* RET
 08-Jan-80
GRAPH .BAK     2  19-Nov-79    GRAPH .FOR     2  08-Jan-80
GRAPH .SAV    18  08-Jan-80    GRAPH .OBJ    16  08-Jan-80
GRAPH .LST     8  08-Jan-80
 5 Files, 46 Blocks
 797 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. Thus, 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 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 from the system volume (that is, all the GRAPH files):

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:
 08-Jan-80
SUM    .MAC     3  08-Jan-80   EXAMP .MAC     3  19-Nov-79
GRAPH  .FOR     2  08-Jan-80   EXAMP .FOR     2  19-Nov-79
GRAPH  .SAV    18  08-Jan-80   GRAPH .LST     8  08-Jan-80
GRAPH  .OBJ    16  08-Jan-80
 7 Files, 52 blocks
 4710 Free blocks
```

This completes the FORTRAN 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 (DEC-11-LFLRA-C-D, DN1). Maynard, Mass.: Digital Equipment Corporation, 1977.

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

RT-11 FORTRAN IV Installation Guide (AA-5240B-TC). Maynard, Mass.: Digital Equipment Corporation, 1978.

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

RT-11 RSTS-E FORTRAN IV User's Guide (DEC-11-LRRUB-A-D). Maynard, Mass.: Digital Equipment Corporation, 1977.

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[1] 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 like a calculator to obtain instant answers to mathematical problems.

THE BASIC–11 PROGRAMMING LANGUAGE

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

You write a BASIC 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 language statements that define the operations to be performed.

When you are satisfied with the logic of your BASIC source program, you create it as a file. However, unlike other programming languages that you may use, you create the file under the control of the BASIC 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 language processor to create and edit the program, list it, run it, and save it for later use.

THE BASIC LANGUAGE PROCESSOR

The BASIC 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 a message on the terminal 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.

---

[1]BASIC–11 is a superset of the standard BASIC language developed at Dartmouth College.

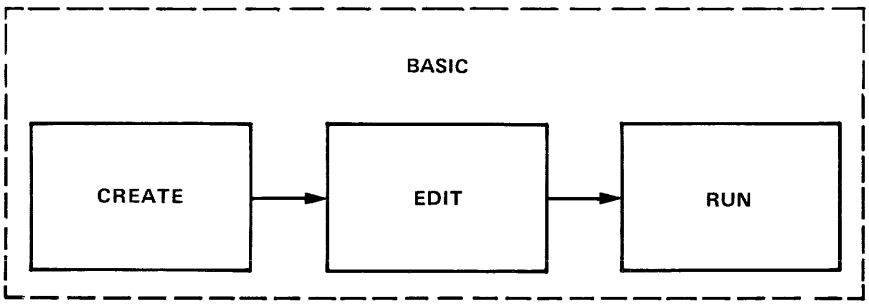


Figure 10-1 Functions of the BASIC Language Processor

USING THE BASIC INTERPRETER

The functions of program creation, editing, processing, and execution are all handled by the BASIC language processor. Some RT-11 systems store the BASIC interpreter (language processor) on a volume apart from the system volume.[1] You can quickly determine whether the BASIC 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 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 interpreter:

Long and Short Command Format

```
.BASIC RET
BASIC-11/RT-11 V02-03B
OPTIONAL FUNCTIONS (ALL, NONE, OR INDIVIDUAL)?
```

A prompting message is printed by BASIC. 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. BASIC's functions include opera-

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

tions that calculate random numbers, determine 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. However, for now, instruct BASIC to save all the functions by typing:

```
A(RET)
READY
```

BASIC prints the READY message to indicate that it is ready to accept a BASIC command. Any text that you type that is not preceded by a BASIC 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 execution that is completed or interrupted by a double CTRL/C, or after any BASIC wait condition that is terminated by a single CTRL/C.

BYE

NOTE

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

Immediate Mode

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

```
PRINT (128+75)*3 (RET)
609
```

BASIC 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. As another example:

PRINT

```
PRINT INT(34.67) (RET)
 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 considers all the information, calculates the answer and prints it on the terminal. For example:

```
A=5\B=14\C=.3729RET

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 program.

Creating and Editing a BASIC Program

To create a BASIC program, simply 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 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 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 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
```

or by using BASIC's DEL command[1]. Use the DEL command (DELETE) to erase a single line or several:

DEL

```
DEL 15-20RET
```

This erases all numbered statement lines with numbers between, including lines 15 and 20.

To list lines of a program, BASIC 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
```

[1]Do not confuse the BASIC DEL command with the DELETE key on the terminal keyboard.

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

```
LIST 5 (RET)

NONAME     08-JUL-77   00:18:49

5 FOR I=1 TO 10

READY
```

LISTNH

Notice that BASIC prints a header line. Since you have not as yet assigned a name to your program, BASIC 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. Type:

```
SCR (RET)

READY
```

All program lines are erased from memory.

SUMMARY: BASIC 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[1], using the appropriate BASIC 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 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 MAY TAKE"
115 PRINT "1, 2, OR 3 MATCHES. TYPE YOUR CHOICE FOLLOWED BY A CAR-"
120 PRINT "RIAGE RETURN. THEN THE COMPUTER CHOOSES 1, 2, OR 3"
125 PRINT "MATCHES. YOU CHOOSE AGAIN, AND SO ON. WHOEVER MUST"
130 PRINT "TAKE THE LAST MATCH, LOSES."
140 PRINT \ LET M=23
```

[1]23 Matches, 101 BASIC Computer Games, Maynard, Mass.: Digital Equipment Corporation, 1975.

```
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
```

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 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 illegal 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 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 RUN command initiates program execution. This command prints a header that includes the program name, data, and time. If you want to omit the header line, type the RUNNH commmand 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 only, returns you to BASIC 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 (RET)
?SYNTAX ERROR AT LINE 250

READY
```

BASIC discovered an error[1] in line 250 that prevents further processing. Check line 250 in your listing or list it on the terminal:

```
250 IF H<>INT(H THEN 510

READY
```

[1]Refer to the *RT-11 System Message Manual* for greater detail about any messages printed during normal system use.

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)@ (RET)
250 IF H<>INT(H) THEN 510

READY
```

You are ready to run the program again. Type:

```
RUNNH (RET)
```

BASIC 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?
```

CTRL/C CTRL/C

Type your choice again. But notice this time that a different kind of error is detected. The BASIC 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 command mode.

```
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 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 does not print an error message. Instead BASIC 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 (RET)

THE COMPUTER TOOK 1 ....
THERE ARE NOW 21 MATCHES.

HOW MANY DO YOU TAKE? 1 (RET)

THE COMPUTER TOOK 3 ....
THERE ARE NOW 17 MATCHES.

HOW MANY DO YOU TAKE? 2 (RET)

THE COMPUTER TOOK 2 ....
THERE ARE NOW 13 MATCHES.
```

```
HOW MANY DO YOU TAKE? 1 (RET)

THE COMPUTER TOOK 3 ....
THERE ARE NOW 9 MATCHES.

HOW MANY DO YOU TAKE? 1 (RET)

THE COMPUTER TOOK 3 ....
THERE ARE NOW 5 MATCHES.

HOW MANY DO YOU TAKE? 3 (RET)

THE COMPUTER TOOK 1 ....
THERE ARE NOW 1 MATCHES.

HOW MANY DO YOU TAKE? 0 (RET)
ENTER ONLY 1, 2, OR 3.

HOW MANY DO YOU TAKE? 1 (RET)

THE COMPUTER WON.

READY
```

SUMMARY: BASIC EXECUTION COMMANDS

RUN
: Execute the BASIC program currently in memory; print a header line including the program name, date, and version number.

RUNNH
: Execute the BASIC program currently in memory; omit the header line.

CTRL/C
: Under control of BASIC only, interrupt execution of the BASIC program and return control to BASIC command mode.

BYE
: Return control to monitor command mode (only when using BASIC).

FILE MAINTENANCE

You can transfer the BASIC 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 giving it 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.

Use the SAVE command to store this BASIC program as MATCH.BAS on the storage volume (VOL:) as follows:

SAVE

```
SAVE VOL:MATCH RET

READY
```

After you save a BASIC program on a storage volume, you can create a new program in memory by typing the BASIC NEW command. This command erases the current memory contents and asks you for a new program name:

NEW

```
NEW RET
NEW FILE NAME--
```

Type any file name you wish and BASIC assigns it to the file you create. Or you can respond by typing only a carriage return; BASIC then assigns the file name NONAME.

Another way to create a new program in memory is to type the BASIC SCR command. This command simply erases the current memory contents. It assigns the name NONAME:

```
SCR RET

READY
```

To use an existing BASIC program, one that you have previously stored on a storage volume, type the BASIC OLD command:

OLD

```
OLD RET
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 assumes DK: (the default volume); if you omit an explicit file type, BASIC assumes .BAS. BASIC erases memory and then copies the program from the volume into memory. For example, type:

```
VOL:MATCH RET
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 REPLACE command:

```
REPLACE VOL:MATCH (RET)

READY
```

REPLACE

The REPLACE command replaces an existing file with a new version.

The SAVE and REPLACE commands copy a BASIC program from computer memory to a storage volume. As these commands copy the program, they convert it from the internal format used by BASIC to ASCII format. Thus, you can, if you prefer, use the RT-11 editor to create and edit BASIC programs, since the editor also uses ASCII format. However, many users would rather use BASIC to create and edit a BASIC program, since they can then run the program, reedit it, rerun it, and save the new version — all in BASIC 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 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-80
SUM    .MAC     EXAMP .MAC     GRAPH .FOR     EXAMP .FOR     GRAPH .SAV
MATCH .BAS     GRAPH .LST     GRAPH .OBJ
 8 Files, 55 Blocks
 4707 Free blocks
```

SUMMARY: BASIC FILE MAINTENANCE COMMANDS

NEW

Create a new BASIC program, assigning the file name indicated.

OLD

Copy into memory (for use under BASIC) an existing BASIC program.

REPLACE
Copy the BASIC program currently in memory to the indicated storage volume, replacing the version that already exists on that volume.

SAVE
Copy the BASIC program currently in memory to the indicated storage volume.

This completes the BASIC 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 (DEC-11-LIBBB-A-D). Maynard, Mass.: Digital Equipment Corporation, 1976.

A reference manual and guide to programming in the BASIC-11 language.

BASIC-11/RT-11 Installation Guide (DEC-11-LIBTA-A-D). Maynard, Mass.: Digital Equipment Corporation, 1977.

An RT-11-specific manual that contains instructions for installing the RT-11 BASIC language processor and lists known problems and differences between versions.

BASIC-11/RT-11 User's Guide (DEC-11-LIBUA-A-D,DN1). Maynard, Mass.: Digital Equipment Corporation, 1978.

An RT-11-specific manual that contains information necessary to create, edit, run, and debug a BASIC 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 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

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.

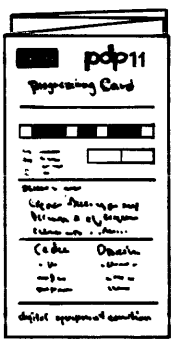


Figure 11-1 PDP-11 Programming Card

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 either 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 make 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

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.

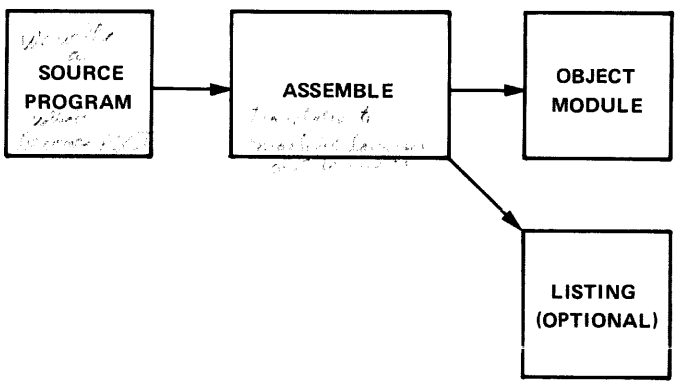


Figure 11-3 Function of a MACRO-11 Assembler

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

The program counter keeps track of addresses in computer memory where instructions and data will be stored.

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 16,384 bytes (8,192 words); most RT-11 systems have more than that number.

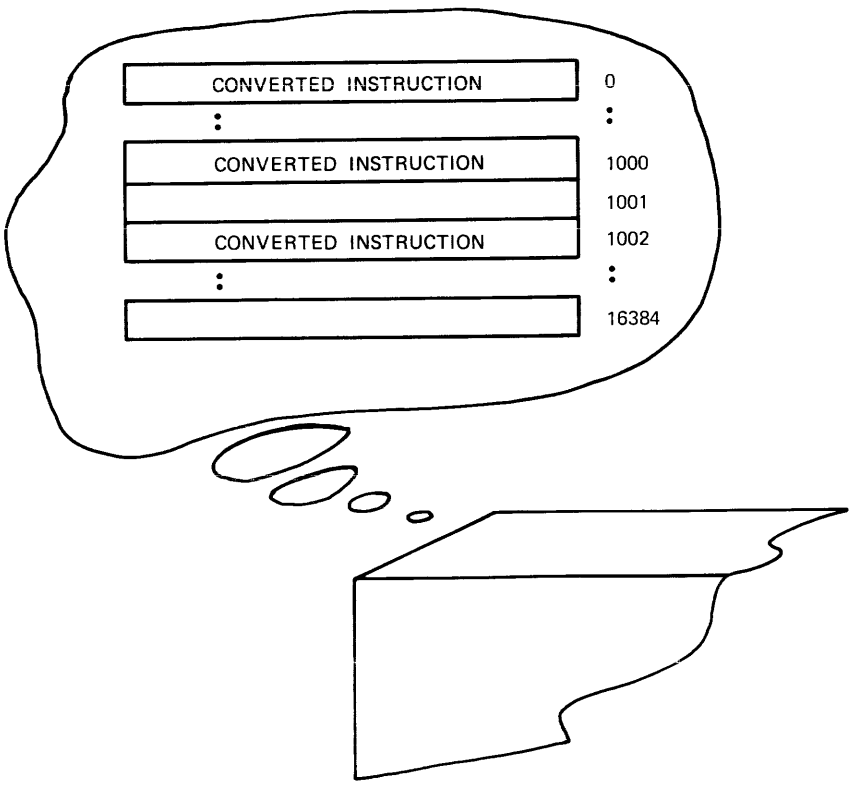


Figure 11-4 PDP-11 Computer Memory

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.[1] 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
.	.
.	.

[1]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:

```
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).

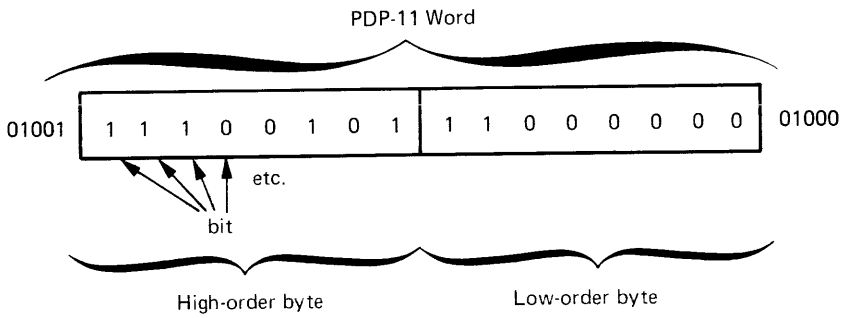


Figure 11–5 PDP–11 Word

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):

```
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 it to object code.

Copy the MACRO 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 source programs, the one you created (SUM.MAC) and the one provided for you (EXAMP.MAC). 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 (RET)
From? VOL:SUM.MAC (RET)
To  ? SUM.MAC (RET)
```

Short Command Format

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

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 source program is the monitor MACRO command.

MACRO

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.)

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:

```
?ERRORS DETECTED:  6
```

This message indicates the number of lines in which the assembler detected errors during processing. In this case, the assembler found six lines in your source program with errors. 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)

```
.PRINT SUM.LST (RET)
```

(Terminal)

```
.TYPE SUM.LST (RET)
```

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.

```
 TY SUM.LST
SUM.MAC VERSION 1       MACRO V04.00  26-NOV-79 12:18:37 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
      9                                 ;       'E' = THE SUM OF THE RECIPROCALS OF THE FACTORIALS
     10                                 ;       1/0! + 1/1! + 1/2! + 1/3! + 1/4! + 1/5! + ...
     11
M    12 000000                          EXP:    .PRINT  #MESSAG          ;PRINT INTRODUCTORY TEXT
     13 000006  012705  000106                  MOV     #N,R5            ;NO. OF CHARS OF 'E' TO PRINT
     14 000012  012700  000107          FIRST:  MOV     #N+1,R0          ;NO. OF DIGITS OF ACCURACY
U    15 000016  012701  000000                  MOV     #A,R1            ;ADDRESS OF DIGIT VECTOR
     16 000022  006311                  SECOND: ASL     @R1              ;DO MULTIPLY BY 10 (DECIMAL)
     17 000024  011146                          MOV     @R1,-(SP)        ;SAVE *2
     18 000026  006311                          ASL     @R1              ;*4
     19 000030  006311                          ASL     @R1              ;*8
     20 000032  062621                          ADD     (SP)+,(R1)+      ;NOW *10, POINT TO NEXT DIGIT
     21 000034  005300                          DEC     R0               ;AT END OF DIGITS?
     22 000036  001371                          BNE     SECOND           ;BRANCH IF NOT
     23 000040  012700  000106                  MOV     #N,R0            ;GO THRU ALL PLACES, DIVIDING
     24 000044  014103                  THIRD:  MOV     -(R1),R3         ;BY THE PLACES INDEX
     25 000046  012702  177777                  MOV     #-1,R2           ;INIT QUOTIENT REGISTER
     26 000052  005202                  FOURTH: INC     R2               ;BUMP QUOTIENT
     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                                                                   ;FOR NEXT DIGIT
AK   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 SO
     38                                                                   ;THAT IT IS ONLY 1 DIGIT
     39 000104  103375                          BCC     FIFTH            ;(REALLY DIVIDE BY 10)
     40 000106  062700  000070                  ADD     #10+'0,R0        ;MAKE DIGIT ASCII
U    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
M    47 000124  000107                  EXP:    .REPT   N+1
     48                                         .WORD   1                ;INIT VECTOR TO ALL ONES
     49                                         .ENDR
     50
     51 000342     124     110     105  MESSAG: .ASCII  /THE VALUE OF E IS:/ <15><12> /2./ <200>
        000345     040     126     101
        000350     114     125     105
        000353     040     117     106
        000356     040     105     040
        000361     111     123     072
        000364     015     012     062
SUM.MAC VERSION 1       MACRO V04.00  26-NOV-79 12:18:37 PAGE 1-1

        000367     056     200
     52                                         .EVEN
     53
D    54          000000'                       .END    EXP
SUM.MAC VERSION 1       MACRO V04.00  26-NOV-79 12:18:37 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
        000372     001
ERRORS DETECTED:  6
```

```
VIRTUAL MEMORY USED:  8448 WORDS  ( 33 PAGES)
DYNAMIC MEMORY AVAILABLE FOR  66 PAGES
DK:SUM,DK:SUM/C=DK:SUM


SUM.MAC VERSION 1       MACRO V04.00  26-NOV-79 12:18:37 PAGE S-1
CROSS REFERENCE TABLE (CREF V04.00 )


.TTYON   1-41
A        1-15
EXP      1-12#    1-47#    1-54
FIFTH    1-37#    1-39
FIRST    1-14#    1-44
FOURTH   1-26#    1-28
MESSAG   1-12     1-51#
N        1-7#     1-13     1-14     1-23     1-47
SECOND   1-16#    1-22
THIRD    1-24#    1-35
SUM.MAC VERSION 1       MACRO V04.00  26-NOV-79 12:18:37 PAGE M-1
CROSS REFERENCE TABLE (CREF V04.00 )


.EXIT    1-3#     1-45
.PRINT   1-3#     1-12
.TTYOU   1-3#
SUM.MAC VERSION 1       MACRO V04.00  26-NOV-79 12:18:37 PAGE E-1
CROSS REFERENCE TABLE (CREF V04.00 )


A        1-32
B        1-54
M        1-12     1-47
R        1-32
U        1-15     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[1] 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 19 of the listing:

```
19 000030  006311            ASL     @R1     ;*8
```

[1]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.

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
```

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:

$$1+(1+(1+...+(1+((1+(1/N))/(N-1))/N-2))/.../2)/1)$$

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 very important and 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 by name from any other part of the program 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 SYSMAC.SML. (SML stands for System Macro Library) SYSMAC.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 available services.

You communicate the need for a monitor service by issuing a programmed request in your source program. A programmed re-

quest 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.[1]

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

[1]Refer to the *RT-11 System Message Manual* for greater detail about any system messages printed during normal system use.

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 logically be defined at line 47. Therefore, line 47 should be changed to read:

```
A:      .REPT   N+1
```

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 (illegal 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 illegal 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.

LINKING OBJECT MODULES TOGETHER

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,[1] since all required object modules must be joined before the program can work.

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 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.

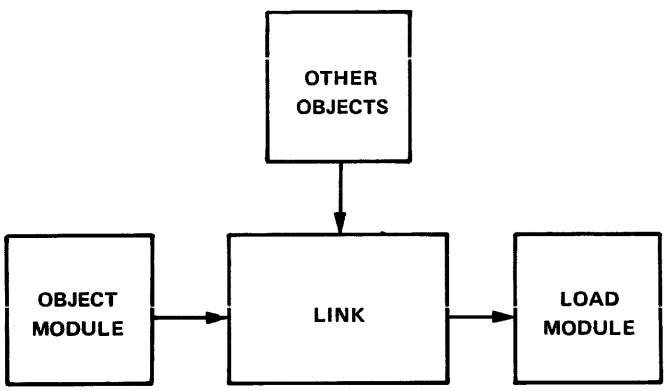


Figure 11-6 The Link Operation

[1]Chapters 12 and 13, respectively, give more information on linking files and using library files.

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.

RUNNING THE MACRO–11 PROGRAM

If your MACRO 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).

To execute the MACRO 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:
2.5/606/606237.2301314.06525/130440275535025.7147773735274474540502.544
.
```

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 very 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, which is the ASCII code for 0) to an ASCII code that can be printed. However, unless you explicitly designate a value as decimal, 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 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 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 pro-

gram. 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 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 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 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 RET
Files? SUM/LIST/CROSSREFERENCE RET
```

Short Command Format

```
.EXECUTE SUM/LIST/CROSSREFERENCE RET
ERRORS DETECTED: 0
THE VALUE OF E IS:
2.5/606/606237.2301314.06525/130440275535025.71477737352744745405O2.544
```

Notice how you use the /LIST and /CROSSREFERENCE options following the file name to request both an assembly and a cross-referenced listing.

SUMMARY: COMMANDS TO RUN MACRO-11 PROGRAMS

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 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.

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.* (RET)
 26-NOV-79
SUM    .SAV  2 26-NOV-79   SUM    .BAK  3 26-NOV-79
SUM    .MAC  3 26-NOV-79   SUM    .OBJ  1 26-NOV-79
SUM    .LST  9 26-NOV-79
 5 Files, 18 Blocks
 2832 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. Thus, 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 (RET)
From? SUM.MAC (RET)
To  ? VOL:SUM.MAC (RET)
```

Short Command Format

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

Similarly, transfer SUM.SAV and SUM.OBJ to your storage volume. This allows you to rerun the MACRO program without reassembling and relinking the source.

Long Command Format

```
.COPY RET
From? SUM.SAV,SUM.OBJ RET
To?  VOL: RET
 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: RET
 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/NOQUERY  RET
Files? SUM.*  RET
```

Short Command Format

```
.DELETE/NOQUERY SUM.*  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 Formats

```
.DIRECTORY VOL: RET
 01-Dec-77
GRAPH .FOR     2  20-NOV-79      SUM   .MAC     3  26-NOV-79
EXAMP .FOR     2  22-OCT-79      EXAMP .MAC     5  22-OCT-79
SUM   .OBJ     1  26-Nov-79      SUM   .SAV     2  26-NOV-79
GRAPH .FOR    10  20-NOV-79      GRAPH .OBJ    14  20-NOV-79
GRAPH .SAV     3  20-NOV-79      MATCH .BAS        20-NOV-79
 10 Files, 61 Blocks
 4701 Free blocks
```

This completes the MACRO demonstration. Continue now to Chapter 12 to learn more about the link operation.

REFERENCES

PDP-11 MACRO-11 Language Reference Manual (AA-5075B-TC). Maynard, Mass.: Digital Equipment Corporation, 1979.

A reference manual for the PDP-11 programmer using the MACRO-11 assembly language.

PDP-11 Peripherals Handbook. Maynard, Mass.: Digital Equipment Corporation, 1978.

A technical description of the PDP-11 peripheral devices, including necessary programming information.

PDP-11 Processor Handbook. Maynard, Mass.: Digital Equipment Corporation, 1978.

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. Maynard, Mass.: Digital Equipment Corporation, 1978.

A general overview and introduction to available PDP-11 software, operation systems, and language processors. See especially Chapters 1, 2, and 3.

RT-11 Programmer's Reference Manual (AA-H378A-TC). Maynard, Mass.: Digital Equipment Corporation, 1980.

An RT-11 system-specific programming manual for the MACRO-11 programmer.

RT-11 System User's Guide (AA-5279B-TC). Maynard, Mass.: Digital Equipment Corporation, 1980.

A guide to the use of the RT-11 operating system. See Chapters 4 and 10.

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 them 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).

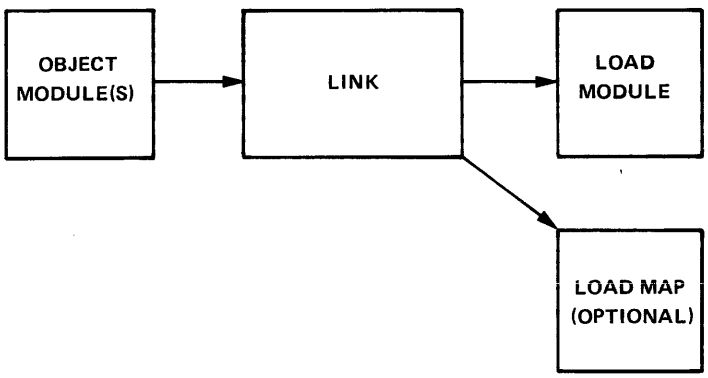


Figure 12–1 Link Functions

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 the appropriate 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 already 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 actually 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 V03.00 5-JUL-77 13:39:00 PAGE 1

      1 000000  000000  000010' 000000          .GLOBL  A,C,VALUE       ;DECLARE A, C, AND VALUE
        000006  000030'
      2                                                                 ;AS GLOBAL SYMBOLS
      3
      4 000010  013500                  A:      MOV     @(R5)+,R0       ;GLOBAL SYMBOL A IS DEFINED
      5                                                                 ;HERE AND CAN BE REFERENCED
      6                                                                 ;FROM OTHER MODULES, PROBABLY
      7                                                                 ;BY A SUBROUTINE CALL
      8 000012  016701  000014                  MOV     LOCAL,R1        ;LOCAL IS AN INTERNAL SYMBOL
      9                                                                 ;DEFINED AND REFERENCED ONLY
     10                                                                 ;WITHIN THIS MODULE
     11
U    12 000016  004767  000000                  USR     PC,C            ;CALL TO GLOBAL ROUTINE C,
     13                                                                 ;DEFINED IN ANOTHER MODULE
     14
     15 000022  013501                          MOV     @(R5)+,R1
     16 000024  005003                          CLR     R3
     17
     18 000026  000207                          RTS     PC
     19
     20 000030  000011                  VALUE:  .WORD   11              ;GLOBAL SYMBOL VALUE IS USED TO
     21                                                                 ;REFERENCE THIS DATA LOCATION
     22
     23 000032  177777                  LOCAL:  .WORD   177777          ;INTERNAL SYMBOL USED FOR DATA
     24         000001                          .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 due to 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.

PROGRAM RELOCATION AND ADDRESS ASSIGNMENT

A second important function of the linker is to "fix" the relative memory addresses so that they are absolute.[1] 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 2, 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 space. So, to

[1] FORTRAN and BASIC 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.

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 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.[1]

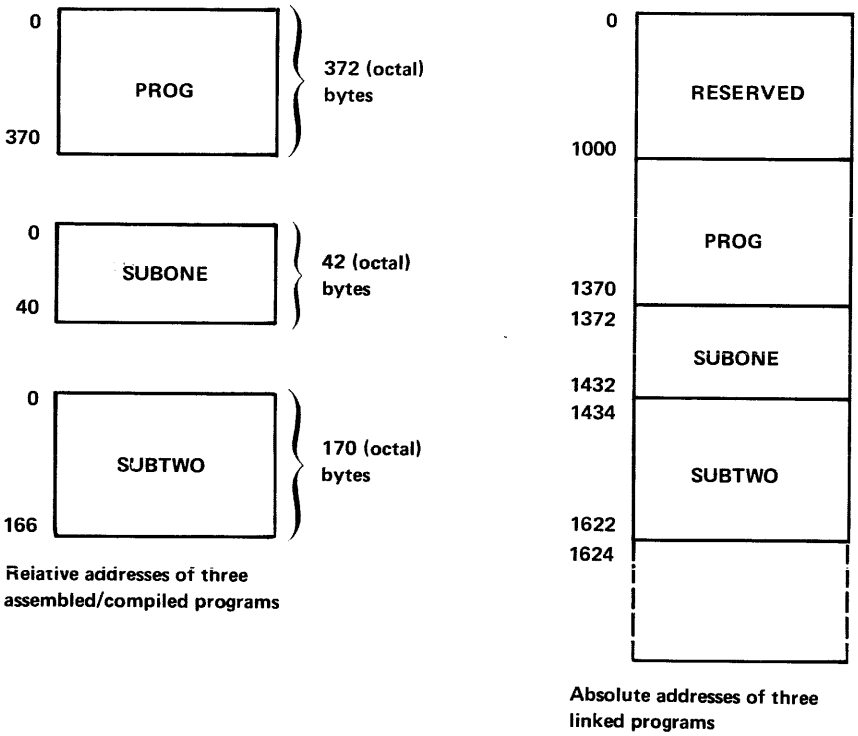


Figure 12-2 Object Module Relocation

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 modularity, and more effective partitioning of memory. The linker processes

[1] A load map for this relocation example is shown later in the chapter.

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 on 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 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 V0 .00 9-JUN-77 15:56:48 PAGE 1

     1
     2                                                          ;UNNAMED RELOCATABLE PROGRAM
     3                                                          ;SECTION IS DECLARED (BY DEFAULT)
     4                                                          ;(".PSECT" IS ASSUMED)
     5
     6
     7
     8 000000  005000                    START:  CLR     R0
     9 000002  012701  000034'                   MOV     #BEG,R1
    10 000006  062100                    LOOP:   ADD     (R1)+,R0
    11 000010  022701  000044'                   CMP     #BEG+10,R1
    12 000014  100374                            BPL     LOOP
    13 000016  012767  002000  000020           MOV     #2000,ADDR
    14 000024  005003                            CLR     R3
    15
    16 000000                                   .PSECT  CLEAR           ;NAMED RELOCATABLE PROGRAM
    17 000000  012703  000100                    MOV     #100,R3         ;SECTION IS DECLARED (VIA ".PSECT NAME")
    18 000004  012701  000044'                   MOV     #ADDR,R1
    19
    20 000010  005021                    AGAIN:  CLR     (R1)+
    21 000012  005303                            DEC     R3
    22 000014  001375                            BNE     AGAIN
    23
    24 000000                                   .ASECT                  ;ABSOLUTE PROGRAM SECTION
    25         000042                           .=42                    ;DECLARED (VIA ".ASECT")
    26 000042  001000                           .WORD   1000            ;THE VALUE 1000 WILL BE
    27                                                                  ;STORED IN ABSOLUTE MEMORY LOCATION 42
    28                                                                  ;WHEN THE PROGRAM IS EXECUTED
    29
    30 000026                                   .PSECT                  ;BACK TO UNNAMED RELOCATABLE
    31 000026  005267  000012                    INC     ADDR           ;PROGRAM SECTION
    32 000032  000000                            HALT
    33 000034  000001  000002  000003  BEG:     .WORD   1,2,3,4
       000042  000004
    34
    35 000044  000000                    ADDR:   .WORD   0
    36                                  ;NOTE THAT YOU CAN WRITE LANGUAGE STATEMENTS THAT WILL BE LOADED
    37                                  ;CONTIGUOUSLY IN MEMORY, BUT DO NOT NECESSARILY OCCUR CONTIGUOUSLY
    38                                  ;IN THE SOURCE PROGRAM (I.E., THE CODE AT LINES 1 - 15AND 29 - 40)
    39
    40
    41         000001                           .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 capable also 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 User's Guide* explains the overlay feature in more detail in Chapter 11. You need not specify an overlay structure for the examples demonstrated in this chapter.

PRODUCING A LOAD MODULE AND A LOAD MAP

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.

/MAP

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 or MACRO object module that you stored on VOL: and use the /MAP option to produce a load map.[1] 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.

Long Command Format

(MACRO object module)

```
.LINK (RET)
Files? VOL:SUM/MAP (RET)
```

(FORTRAN object module)

```
.LINK (RET)
Files? VOL:GRAPH/MAP (RET)
```

Short Command Format

(MACRO object module)

```
.LINK VOL:SUM/MAP (RET)
```

[1]FORTRAN 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 users who have a special FORTRAN language volume, but not a LINK volume, should make sure that the FORTRAN volume is now loaded in device unit 0.

(FORTRAN object module)

```
.LINK VOL:GRAPH/MAP (RET)
```

Now list the .MAP file on either the line printer or terminal, choosing the appropriate command:

Long Command Format

(Line printer) (Terminal)

(MACRO object module)

```
.PRINT (RET)                    .TYPE (RET)
Files? SUM.MAP (RET)            Files? SUM.MAP (RET)
```

(FORTRAN object module)

```
.PRINT (RET)                    .TYPE (RET)
Files? GRAPH.MAP (RET)          Files? GRAPH.MAP (RET)
```

Short Command Format

(Line printer) (Terminal)

(MACRO object module)

```
.PRINT SUM.MAP (RET)            .TYPE SUM.MAP(RET)
```

(FORTRAN object module)

```
.PRINT GRAPH.MAP (RET)          .TYPE GRAPH.MAP (RET)
```

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  V06.01        Load Map        Tue 08-JAN-80 00:51:32
SUM   .SAV        Title:  SUM.MA  Ident:

Section   Addr   Size    Global  Value   Global  Value   Global  Value

. ABS.    000000 001000   (RW,I,GBL,ABS,OVR)
          001000 000372   (RW,I,LCL,REL,CON)

Transfer address = 001000, High limit = 001370 = 380.    words

RT-11 LINK  V06.01        Load Map        Tue 08-JAN-80 14:00:47
GRAPH .SAV        Title:  .MAIN.  Ident:  FORV02
Section  Addr   Size    Global  Value   Global  Value   Global  Value

. ABS.   000000 001000   (RW,I,GBL,ABS,OVR)
                         $USRSW  000000  $RF2A1  000000  $HRDWR  000002
                         $NLCHN  000006  $SYSV$  000011  $WASIZ  000131
                         $LRECL  000210  $TRACE  004737  $EAE    177304
OTS$I    001000 014362   (RW,I,LCL,REL,CON)
                         $$OTSI  001000  $OTI    001026  $$OTI   001030
                         $$SET   002536  $CVTIF  003032  $CVTIC  003046
                         $CVTID  003046  CCI$    003060  CDI$    003060
                         $IC     003060  $ID     003060  CFI$    003074
                         $IR     003074  EXP     003160  SQRT    003520
                         MUF$PS  003714  MUF$MS  003720  MUF$IS  003732
                         $MULF   003740  MUF$SS  003752  $MLR    003752
```

```
                                DIF$PS  004350  DIF$MS  004354  DIF$IS  004366
                                $DIVF   004374  DIF$SS  004406  $DVR    004406
                                ADF$IS  005012  ADF$PS  005020  SUF$PS  005024
                                SUF$MS  005030  ADF$MS  005042  SUF$IS  005052
                                $ADDF   005060  $SUBF   005074  SUF$SS  005106
                                $SBR    005106  ADF$SS  005112  $ADR    005112
                                ADD$    005126  IDINT   005552  INT     005552
                                MAX0    005600  MIN0    005624  CAI$    005650
                                CAL$    005656  ISN$    005706  $ISNTR  005712
                                LSN$    005726  $LSNTR  005732  MOI$IP  006066
                                MOI$SP  006070  MOI$PP  006076  MOI$MP  006102
                                MOI$PS  006112  MOI$PM  006120  MOI$PA  006126
                                MOI$0P  006134  MOI$1P  006142  MOF$SS  006152
                                MOF$SM  006164  MOF$SP  006174  LLE$    006200
                                LEQ$    006202  LGT$    006210  LGE$    006212
                                LNE$    006222  LLT$    006224  MOL$SM  006230
                                MOL$SA  006234  MOL$MS  006240  MOL$MM  006250
                                MOL$MA  006254  MOL$SP  006260  MOL$PP  006266
                                MOL$MP  006272  MOL$PM  006302  MOL$PS  006310
                                MOL$PA  006314  MOL$IM  006322  MOL$IA  006330
                                MOL$IP  006336  STK$L   006346  STK$I   006352
                                STK$F   006356  MOI$RS  006366  MOL$RS  006366
                                MOI$RM  006372  MOI$RP  006376  MOI$RA  006400
                                NGD$S   006404  NGF$S   006404  NGD$M   006416
                                NGF$M   006416  NGD$P   006432  NGF$P   006432
                                NGD$A   006436  NGF$A   006436  ADI$SS  006442
                                ADI$SA  006446  ADI$SM  006452  ADI$IS  006456
                                ADI$IA  006462  ADI$IM  006466  ADI$MS  006472
                                ADI$MA  006476  ADI$MM  006502  SUI$SS  006506
                                SUI$SA  006512  SUI$SM  006516  SUI$IS  006522
                                SUI$IA  006526  SUI$IM  006532  SUI$MS  006536
                                SUI$MA  006542  SUI$MM  006546  ICI$S   006552
                                ICI$M   006556  ICI$P   006562  ICI$A   006564
                                DCI$S   006570  DCI$M   006574  DCI$P   006600
                                DCI$A   006602  CMI$SS  006606  CMI$SI  006612
                                CMI$SM  006616  CMI$IS  006622  CMI$II  006626
                                CMI$IM  006632  CMI$MS  006636  CMI$MI  006642
                                CMI$MM  006646  NMI$1M  006652  NMI$1I  006664
                                BLE$    006674  BEQ$    006676  BGT$    006704
                                BGE$    006706  BRA$    006710  BNE$    006714
                                BLT$    006716  MOF$RS  006726  MOF$RM  006734
                                MOF$RA  006744  MOF$RP  006750  MOF$MS  006754
                                MOF$PS  006766  MOF$MM  006772  MOF$MA  007004
                                MOF$MP  007012  MOF$PM  007020  MOF$PA  007024
                                MOF$PP  007030  IOR$    007034  AND$    007040
                                EQV$    007046  XOR$    007050  TSL$S   007064
                                TSL$M   007070  TSL$I   007074  TSL$P   007102
                                $OTIS   007110  $$OTIS  007112  RET$L   007232
                                RET$F   007236  RET$I   007244  RET$    007246
                                MOI$SS  007302  MOL$SS  007302  MOI$SM  007306
                                MOI$SA  007312  MOI$IS  007316  MOL$IS  007316
                                REL$    007316  MOI$IM  007322  MOI$IA  007326
                                MOI$MS  007332  MOI$MM  007336  MOI$MA  007342
                                MOI$0S  007346  MOI$0M  007352  MOI$0A  007356
                                MOI$1S  007362  MOI$1M  007370  MOI$1A  007376
                                EXIT    007404  SAL$IM  007410  SAL$SM  007412
                                SVL$IM  007420  SVL$SM  007422  SAL$MM  007430
                                SVL$MM  007434  $CVTFB   007440  $CVTFI  007440
                                $CVTCB  007454  $CVTCI  007454  $CVTDB  007454
                                $CVTDI  007454  CIC$    007466  CID$    007466
                                CLC$    007466  CLD$    007466  $DI     007466
                                CIF$    007476  CLF$    007476  $RI     007476
                                CIL$    007620  CLI$    007624  $ERRTB  007626
                                $SHORT  007626  $ERRS   007733  TVL$    010254
                                $TVL    010254  TVF$    010262  $TVF    010262
                                TVD$    010270  $TVD    010270  TVQ$    010276
                                $TVQ    010276  TVP$    010304  $TVP    010304
                                TVI$    010312  $TVI    010312  $STPS   010446
                                STP$    010454  $STP    010454  FOO$    010460
                                $EXIT   010500  END$    010624  ERR$    010636
                                $END    010650  $ERR    010666  IFW$    010710
                                $IFW    010714  IFW$$   010756  $CHKER  011026
                                $IOEXI  011052  $EOL    011100  EOL$    011102
                                $VRINT  011216  SAL$IP  011220  SAL$SP  011222
                                SVL$IP  011230  SVL$SP  011232  SAL$MP  011240
                                SVL$MP  011244  SAI$IM  011250  SAI$SM  011252
                                $ROUND  011256  SVI$IM  011302  SVI$SM  011304
                                SAI$MM  011314  SVI$MM  011320  SAVRG$  011324
                                THRD$   011502  $PUTRE  011504  $WAIT   012012
                                $FCHNL  012054  $INITI  012152  $CLOSE  012264
                                $PUTBL  012730  $GETBL  013140  $EOFIL  013324
                                $EOF2   013340  $FIO    014100  $$FIO   014104
                                $DUMPL  015234
OTS$P   015362 000050             (RW,D,GBL,REL,OVR)
SYS$I   015432 000244             (RW,I,LCL,REL,CON)
                                LEN     015432  REPEAT  015450  SCOPY   015602
USER$I  015676 000000             (RW,I,LCL,REL,CON)
$CODE   015676 001336             (RW,I,LCL,REL,CON)
                                $$OTSC  015676  FUN     016574  PUTSTR  016742
OTS$D   017234 001010             (RW,I,LCL,REL,CON)
                                $$OTSO  017234  $OPEN   017234
SYS$D   020244 000000             (RW,I,LCL,REL,CON)
$DATAP  020244 000106             (RW,D,LCL,REL,CON)
```

```
 OTS$D    020352 000006   (RW,D,LCL,REL,CON)
 OTS$S    020360 000002   (RW,D,LCL,REL,CON)
                         $AOTS   020360
 SYS$S    020362 000004   (RW,D,LCL,REL,CON)
                         $SYSLB  020362  $LOCK   020364  $CRASH  020365
 $DATA    020366 000542   (RW,D,LCL,REL,CON)
 USER$D   021130 000000   (RW,D,LCL,REL,CON)
 .$$$$.   021130 000000   (RW,D,GBL,REL,OVR)

Transfer address = 015676, High limit = 021126 = 4395.  words
RT-11 LINK  V06.01       Load Map                TUE 08-JAN-80 00:20:09
                  TITLE:  TEST    INDENT: V00.00

SECTION   ADDR   SIZE    GLOBAL  VALUE   GLOBAL  VALUE   GLOBAL  VALUE

. ABS     000000 001000   (RW,I,GRL,ARS,OVR)
          001000 000624   (RW,I,LCL,REL,CON)
                         PROG    001000  SURONE  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 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 load map, remembering that it reflects the appropriate expansions into machine language code provided by the FORTRAN 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 OTSP, SYSI, and $CODE. Global symbols and their respective addresses appear to the right of all sections. The total number of bytes used is 21126, or 4395 (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 624 bytes long. This program section consists of a main program, caller 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. They are not generally obtained or used for FORTRAN 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 demonstration program.

SUMMARY: COMMANDS FOR LINKING PROGRAMS

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

FORTRAN 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.

REFERENCE

RT-11 System User's Guide (AA-5279B-TC). Maynard, Mass.: Digital Equipment Corporation, 1980.

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

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.

KINDS OF LIBRARY FILES

There are two kinds of library files — macro libraries and object libraries.

Macro Libraries

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.

Object 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 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 and FORTRAN programs; the routine in the third module can be used only by FORTRAN 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 users should create all three files; MACRO users (who do not use FORTRAN) 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 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 programs and writes a variable-length character string on a specified FORTRAN 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 users who are not familiar with the assembly process simply type the MACRO commands as shown.

Long Command Format

```
.MACRO RET
Files? FIRST RET
ERRORS DETECTED:  0
```

Short Command Format

```
.MACRO FIRST RET
ERRORS DETECTED:  0
```

The command creates an object module called FIRST.OBJ on the system volume. The assembler prints a message on the terminal, indicating the number of errors encountered during assembly. This message should show 0 errors.

In the same way, assemble SECOND.MAC. Again, no errors should occur.

Long Command Format

```
.MACRO(RET)
Files? SECOND(RET)
ERRORS DETECTED:  0
```

Short Command Format

```
.MACRO SECOND(RET)
ERRORS DETECTED:  0
```

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 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(RET)
Files? THIRD(RET)
PUTSTR
```

Short Command Format

```
.FORTRAN THIRD(RET)
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.

Building the Object Library

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.

LIBRARY/CREATE

Long Command Format

```
.LIBRARY/CREATE(RET)
Library? LIBRA(RET)
Files  ? FIRST,SECOND(RET)
```

Short Command Format

```
.LIBRARY/CREATE LIBRA FIRST,SECOND(RET)
```

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 code to the LIST option that follows.

LIBRARY/LIST

Long Command Format

(Line printer)

```
.LIBRARY/LIST(RET)
Library? LIBRA(RET)
```

(Terminal)

```
.LIBRARY/LIST:TT:(RET)
Library? LIBRA(RET)
```

Short Command Format

(Line printer)

```
.LIBRARY/LIST LIBRA(RET)
```

(Terminal)

```
.LIBRARY/LIST:TT: LIBRA(RET)
```

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 V03.10 TUE 11-JAN-80 11:03:29
DK:LIBRA.OBJ           TUE 11-JAN-80 10:59:43

MODULE      GLOBALS      GLOBALS      GLOBALS

            LEN          PRINT
            ITTOUR
```

Updating the Object Library

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.

LIBRARY/ INSERT

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/INSERT(RET)
Library? LIBRA(RET)
Files  ? THIRD(RET)
```

Short Command Format

```
.LIBRARY/INSERT LIBRA THIRD(RET)
```

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 V03.10 TUE 11-JAN-80 11:05:1
DK:LIBRA.OBJ           TUE 11-JAN-80 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.

LIBRARY/ REMOVE

You can remove individual globals by using the REMOVE option. For example, to remove the global ITTOUR, type:

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 V03.10 TUE 11-JAN-80 11:10:22
DK:LIBRA.OBJ           TUE 11-JAN-80 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 User's Guide*, Chapter 12.

SUMMARY: COMMANDS FOR MAINTAINING LIBRARY FILES

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.

FILE MAINTENANCE

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 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 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 User's Guide (AA-5279B-TC), Maynard, Mass.: Digital Equipment Corporation, 1980.

A guide to the use of the RT-11 operating system. See Chapter 12.

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 actually writing the program code. 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.

AVOIDING PROGRAMMING ERRORS

There are several steps you can take 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 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 sepa-

rately 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 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'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.

FORTRAN 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, FORTRAN 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, FORTRAN IV has an optional interactive debugger called FDT (FORTRAN 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 FORTRAN IV users who will be combining MACRO and FORTRAN 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.

USING THE ON-LINE DEBUGGING TECHNIQUE

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 program that needs debugging. You then start execution of the resulting load module, not at the transfer address of your program, but at the entry point of the ODT module (shown on the linker load map as the global symbol O.ODT). Once ODT is started, you can use its special debugging commands to control the execution of your assembled machine language program from the console terminal, to examine memory locations, to change their contents, and to stop and continue program execution.

The MACRO demonstration program in Chapter 11 still contains one error, which you can locate and correct using ODT. Several ODT debugging commands are demonstrated in the process.

Throughout the examples in this chapter you need to refer to the program assembly listing that you produced in Chapter 11 (SUM) and stored on the storage volume. Print it now on either the terminal or line printer:

Long Command Format

(Line printer)

```
.PRINT RET
Files? VOL:SUM.LST RET
```

(Terminal)

```
.TYPE RET
Files? VOL:SUM.LST RET
```

Short Command Format

(Line printer)

```
.PRINT VOL:SUM.LST RET
```

(Terminal)

```
.TYPE VOL:SUM.LST RET
```

```
SUM.MAC  VERSION 1      MACRO V04.00  8-JAN-80 00:40:00 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
      9                                ;    'E' = THE SUM OF THE RECIPROCALS OF THE FACTORIALS
     10                                ;    1/0! + 1/1! + 1/2! + 1/3! + 1/4! + 1/5! + ...
     11
M    12 000000                        EXP:  .PRINT  #MESSAG         ;PRINT INTRODUCTORY TEXT
     13 000006  012705  000106               MOV    #N,R5           ;NO. OF CHARS OF 'E' TO PRINT
     14 000012  012700  000107        FIRST: MOV    #N+1,R0         ;NO. OF DIGITS OF ACCURACY
U    15 000016  012701  000000               MOV    #A,R1           ;ADDRESS OF DIGIT VECTOR
     16 000022  006311                SECOND: ASL    @R1             ;DO MULTIPLY BY 10 (DECIMAL)
     17 000024  011146                       MOV    @R1,-(SP)       ;SAVE *2
     18 000026  006311                       ASL    @R1             ;*4
     19 000030  006311                       ASL    @R1             ;*8
     20 000032  062621                       ADD    (SP)+,(R1)+     ;NOW *10, POINT TO NEXT DIGIT
     21 000034  005300                       DEC    R0              ;AT END OF DIGITS?
     22 000036  001371                       BNE    SECOND          ;BRANCH IF NOT
     23 000040  012700  000106               MOV    #N,R0           ;GO THRU ALL PLACES, DIVIDING
     24 000044  014103                THIRD: MOV    -(R1),R3        ;BY THE PLACES INDEX
```

```
     25 000046  012702  177777                MOV     #-1,R2          ;INIT QUOTIENT REGISTER
     26 000052  005202                FOURTH: INC     R2              ;BUMP QUOTIENT
     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                                                               ;FOR NEXT DIGIT
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 SO
     38                                                               ;THAT IT IS ONLY 1 DIGIT
     39 000104  103375                        BCC     FIFTH           ;(REALLY DIVIDE BY 10)
     40 000106  062700  000070                ADD     #10+'0,R0       ;MAKE DIGIT ASCII
U    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
M    47 000124  000107                EXP:    .REPT   N+1
     48                                       .WORD   1               ;INIT VECTOR TO ALL ONES
     49                                       .ENDR
     50
     51 000342     124     110     105  MESSAG: .ASCII  /THE VALUE OF E IS:/ <15><12> /2./ <200>
        000345     040     126     101
        000350     114     125     105
        000353     040     117     106
        000356     040     105     040
        000361     111     123     072
        000364     015     012     062

SUM.MAC  VERSION 1      MACRO V04.00  8-JAN-80 00:40:00 PAGE 1-1


        000367     056     200
     52                                  .EVEN
     53
D    54          000000'                .END    EXP

SUM.MAC  VERSION 1      MACRO V04.00  8-JAN-80 00:40:00 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
        000372     001
ERRORS DETECTED:  6

VIRTUAL MEMORY USED:  8448 WORDS  ( 33 PAGES)
DYNAMIC MEMORY AVAILABLE FOR  66 PAGES
DK:SUM,DK:SUM/C=DK:SUM

SUM.MAC  VERSION 1      MACRO V04.00  8-JAN-80 00:40:00 PAGE S-1
CROSS REFERENCE TABLE (CREF V04.00 )

.TTYON   1-41
A        1-15
EXP      1-12#    1-47#    1-54
FIFTH    1-37#    1-39
FIRST    1-14#    1-44
FOURTH   1-26#    1-28
MESSAG   1-12     1-51#
N        1-7#     1-13     1-14     1-23     1-47
SECOND   1-16#    1-22
THIRD    1-24#    1-35

SUM.MAC  VERSION 1      MACRO V04.00  8-JAN-80 00:40:00 PAGE M-1
CROSS REFERENCE TABLE (CREF V04.00 )

.EXIT    1-3#     1-45
.PRINT   1-3#     1-12
.TTYOU   1-3#

SUM.MAC  VERSION 1      MACRO V04.00  8-JAN-80 00:40:00 PAGE E-1
CROSS REFERENCE TABLE (CREF V04.00 )

A        1-32
D        1-54
M        1-12     1-47
R        1-32
U        1-15     1-41
```

Now link the MACRO program object module (SUM.OBJ) stored on the storage volume (VOL:) with ODT.OBJ by using the /DEBUG option, and print a load map directly on the terminal or line printer, choosing one of the following commands:

LINK/DEBUG

Long Command Format

(Line printer)

```
.LINK/MAP/DEBUG(RET)
Files? VOL:SUM(RET)
```

(Terminal)

```
.LINK/MAP:TT:/DEBUG(RET)
Files? VOL:SUM(RET)
```

Short Command Format

(Line printer)

```
.LINK/MAP/DEBUG VOL:SUM(RET)

    RT-11 LINK  V06.01       Load Map           Fri 11-Jan-80 13:11:26
    SUM   .SAV      Title:   ODT       Ident:   V04.00

    Section  Addr   Size     Global    Value    Global  Value    Global  Value

     . ABS.  000000 001000   (RW,I,GBL,ABS,OVR)
             001000 000372   (RW,I,LCL,REL,CON)
    $ODT$    001372 006152   (RW,I,LCL,REL,CON)
                             O.ODT   001624

    Transfer address = 001624, High limit = 007542 = 1969. words
```

(Terminal)

```
.LINK/MAP:TT:/DEBUG VOL:SUM(RET)
```

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 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(RET)
 ODT  V04.00
*
```

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.[1] 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 DEL key; ODT responds with a ? and an asterisk, allowing you to enter another command.

[1]Be sure to read Chapter 21 of the *RT-11 System User's Guide* 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. There are steps you can take to prevent this from occurring.

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 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;0R
```

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 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:
2.5/606/606237,2301314,06525/130440275535025,7147773735274474540502,544
.
```

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 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(RET)

 ODT  V04.00
*
```

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
TB0;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 at line 12 of the assembly listing (.PRINT) 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 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 the next several addresses following it by using two new ODT commands, $ and @:

```
*$1/001124 @
0,000124 /000001LF
0,000126 /000001LF
0,000130 /000001LF
0,000132 /000001RET
```

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:

```
*0,124/000012LF
0,000126 /000012LF
0,000130 /000012LF
0,000132 /000012RET
```

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. Type:

```
*0,110;0B
*;P
EB0;0,000110
```

Now examine the contents of register 0.

```
*$0/000065 \065 =5RET
```

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:
2.
.
```

Restart ODT and reset relocation register 0:

```
.RUN SUM RET

ODT    V04.00
*1000;0R
```

Set a breakpoint at location 76 (line 37 in the assembly listing), and start program execution at its beginning:

```
*0,76;0B
*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, and continue program execution; the correct results should print:

```
*;P
THE VALUE OF E IS:
2.71828182845904523536028747135266249775724709369995957496696762772407663
*
```

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[1]

(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.

[1] Only a very few of the available debugging commands have been demonstrated in this chapter. Consult Chapter 21 of the *RT-11 System User's Guide* 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.

FILE MAINTENANCE

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:

```
ADD   #10,+'0,R0          ;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.

REFERENCE

RT-11 System User's Guide (AA-5279B-TC). Maynard, Mass.: Digital Equipment Corporation, 1980.

A guide to the use of the RT-11 operating system. See Chapter 21 for a detailed description of ODT.

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. Thus far, you have used its single-job environment when running the system utility programs and the demonstration programs one at a time. A second environment, called the foreground/background environment, is also available. This environment allows two independent programs to reside in memory at the same time and to execute concurrently.

Because there are different operating environments, there are actually different monitors. You are familiar with the single-job (SJ) monitor. You have used the single-job monitor so far to control the system and to perform the various exercises in this manual.

To run in the foreground/background environment, you activate a second monitor, called the foreground/background (FB) monitor. The FB monitor is simply an extension of the SJ monitor; it is completely compatible with the SJ monitor, but provides extended monitor command operations for controlling a two-job environment.[1]

THE FOREGROUND/ BACKGROUND ENVIRONMENT

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 pro-

[1]The RT–11 operating system also provides a third operating environment called the extended memory environment. This environment is governed by the extended memory (XM) monitor and allows advanced users to utilize up to 128K (words) of memory. See the *RT–11 Software Support Manual* for more information.

gram 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 compiler, the linker, and so forth.

Foreground/background operation requires that you have at least 16K words of computer memory (each 4K equals 4096 words) plus a system clock. Not all RT-11 computer systems support foreground/background operation, since the hardware it requires is optional. To determine if your system can support FB operation, check the Hardware Configuration section in Chapter 2. If you have at least 16K of memory and the system accepts a TIME command, you can use the foreground/background monitor to perform the exercises in this chapter. Otherwise, you do not have the hardware necessary to support an FB environment, and you should skip ahead to Chapter 16.

Running the Foreground/ Background Programs

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. Besides printing the terminal message when used as a single-job exercise, the background job in a foreground/background environment recognizes 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 can be executed in the background besides DEMOBG. 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.

You first edit, assemble, and link the background job under the single-job monitor. Then you boot the FB monitor into memory. After creating the foreground job, you execute both jobs in a foreground/background environment.

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 in a single-job environment, it displays a message on the terminal. It is assumed that you are running the SJ monitor and 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)
ERRORS DETECTED: 0
```

Short Command Format

```
.MACRO DEMOBG/LIST (RET)
ERRORS DETECTED: 0
```

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/C's.

```
(CTRL/C)
(CTRL/C)

^C
^C

.
```

CHANGING MONITORS

Whenever you bootstrap the RT-11 system, it prints a message on the console terminal telling you which monitor has been loaded into memory. The message for the single-job monitor is:

```
RT-11SJ      V04.xx

.
```

BOOT

The single-job monitor is currently in memory. To use the FB environment, you must reboot the system so that the FB monitor is loaded into memory, overwriting the SJ monitor. You use the monitor BOOT command to make this switch.

If you have not entered the date and time, do so before booting the FB monitor. These features remain active throughout the booting procedure if the BOOT command is used. When changing monitors, any logical names you assigned are lost. Thus, you must reassign your storage volume device as VOL.

Long Command Format

```
.BOOT (RET)
Device or file? RT11FB (RET)

RT-11FB     V04.xx
```

Short Command Format

```
.BOOT RT11FB (RET)

RT-11FB     V04.xx
```

Once the system executes the BOOT command, the monitor formerly in memory is no longer active. It is replaced by the alternate monitor. The message printed on the console terminal tells you which monitor has been loaded.[1]

USING THE FB MONITOR

Using the FB monitor is essentially no different from using the SJ monitor. All commands that are legal in the SJ environment are legal in the FB environment; their syntax and use are exactly the same. In addition, programs that you write for the single-job environment can always run as the background job in the FB environment.

Since the FB monitor is actually an extension of the SJ monitor, it provides some commands and programming features that the SJ monitor does not have. These 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.

Communication in a Two-Job Environment

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.

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 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

[1] To reboot the single-job monitor, simply reply to the BOOT command's DEVICE OR FILE? prompt by typing RT11SJ.SYS (RET).

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.

These conventions apply only if two jobs are running simultaneously. If only one job is running, communication is the same as in the single-job environment.

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. Following assembly, the system prints a message on the terminal indicating the number of errors encountered during assembly. This message will show 0 errors.

Long Command Format

```
.MACRO (RET)
Files? DEMOFG/LIST (RET)
ERRORS DETECTED:   0
```

Short Command Format

```
.MACRO DEMOFG/LIST (RET)
ERRORS DETECTED:   0
```

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.[1] 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.

[1]This command option also applies to compiled FORTRAN programs that are to be linked as a foreground job.

Long Command Format

```
.LINK/FOREGROUND (RET)
Files? DEMOFG (RET)
```

Short Command Format

```
.LINK/FOREGROUND DEMOFG (RET)
```

Executing the Foreground and Background Jobs

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-character code from Table 4–2 for the storage volume in place of xx.

Long Command Format

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

Short Command Format

```
.ASSIGN xx: 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.

LOAD

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:

Long Command Format

```
.LOAD (RET)
Device? LP:=F (RET)
```

Short Command Format

```
.LOAD LP:=F (RET)
```

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. Use this command to start the execution of DEMOFG.REL.

Long and Short Command Format

```
.FRUN DEMOFG (RET)

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 (RET)
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. You can use the background of an FB environment in the same way as the SJ environment. First, terminate the background job DEMOBG, using the double CTRL/C command.

```
(CTRL/C)
(CTRL/C)

.
```

Execute a DIRECTORY command in the background to get a listing of all the .OBJ files on the system volume by typing

```
.DIRECTORY *.OBJ (RET)
```

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 DEMOBG (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>
```

Since you are now using only the background of the foreground/background environment, the system is operating like a single-job system.

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-character device codes.

Check to see if the .LST files were produced as a result of this demonstration.

Long and Short Command Form

```
.DIRECTORY *.LST (RET)
 10-Jan-80
DEMOBG.LST                          4   10-JAN-80
DEMOFG.LST                          6   10-JAN-80
 2 Files, 10 Blocks
 988 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, be sure to read the *RT-11 Software Support Manual* for coding restrictions.

SUMMARY: COMMANDS USED IN AN FB ENVIRONMENT

BOOT
Bootstrap the indicated monitor (RT-11SJ, RT-11FB, RT-11XM) on the system volume.

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 dh
Make the indicated device handler (dh) resident in memory.

UNLOAD dh
Make the indicated device handler (dh) nonresident in memory, reclaiming its memory space.

UNLOAD FG
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 on your system volume named DEMOFG.LST. Thus, 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 DEMOBG, 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/NOQUERY (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/NOQUERY 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)
```

REFERENCES

RT-11 Software Support Manual (AA-5280B-TC). Maynard, Mass.: Digital Equipment Corporation, 1980.

A technical manual providing RT-11 programming concepts.

RT-11 System User's Guide (AA-5279B-TC). Maynard, Mass.: Digital Equipment Corporation, 1980.

A guide to the use of the RT-11 operating system. See Chapters 2, 3, and 4.

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.[1]

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.

CREATING 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.

Entering Monitor Commands

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 (RET)
INFIL.MAC (RET)
OUTFIL.MAC (RET)

COPY INFIL.MAC OUTFIL.MAC (RET)
```

[1]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 discussed in the *RT-11 System User's Guide*.)

Since monitor prompts are not included in the indirect file, using the long command format requires that you anticipate each prompt and its proper response. It is suggested that you use the short command format and insert 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.

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` RET `*IDATE 14-JAN-80` RET `TIME 8:00:00` RET	Enter a hypothetical date and time (if your system has a clock).
`DATE` RET	Print the date.
`DEASSIGN` RET	Deassign all previous device assignments and set new ones as follows:
`ASSIGN TT: LP:` RET	Assign the logical name LP: to the terminal.
`ASSIGN XXn VOL:` RET	Assign the logical name VOL: to the storage volume (xx).
`DIRECTORY/BRIEF VOL:` RET	List an abbreviated directory of VOL:.
`COPY VOL:GRAPH.FOR GRAPH.FOR` RET	FORTRAN users insert this command to copy the FORTRAN demo program to the system volume.
`COPY VOL:SUM.MAC SUM.MAC` RET	MACRO users insert this command to copy the MACRO demo program to the system volume.

```
COPY VOL:MATCH.BAS
MATCH.BAS (RET)
```

BASIC users insert this command to copy the BASIC demo program to the system volume.

```
FORTRAN/LIST GRAPH (RET)
LINK/MAP GRAPH (RET)
```

FORTRAN users who do not need to load the language volume include these commands to compile and link the demo program.

```
MACRO/LIST/CROSSREFERENCE SUM (RET)
LINK/MAP SUM (RET)
```

All users assemble and link the demo program.

```
RENAME MATCH.BAS MATCH.MAP (RET)
```

BASIC users simply rename the demo program.

```
MACRO/LIST/CROSSREFERENCE DEMOFG (RET)
LINK/FOREGROUND/MAP DEMOFG (RET)
```

All users assemble and link the DEMOFG file.

```
DIRECTORY *.OBJ (RET)
```

List a directory of object files.

```
DELETE/NOQUERY GRAPH.* (RET)
```

FORTRAN users delete the GRAPH files.

```
DELETE/NOQUERY SUM.* (RET)
```

MACRO users delete the SUM files.

```
DELETE/NOQUERY MATCH.MAP (RET)
```

BASIC users delete the MATCH file.

```
DEASSIGN (RET)
```

Deassign all device assignments.

```
TIME (RET)
```

If your system has a clock, print the time to show how long total processing took.

```
  (ESC) (ESC)
*B/L (ESC) (ESC)
DATE 14-JAN-80
TIME 8:00:00
```

Now 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
```

```
MACRO/LIST/CROSSREFERENCE SUM
LINK/MAP SUM
RENAME MATCH.BAS MATCH.MAP
MACRO/LIST/CROSSREFERENCE DEMOFG
LINK/FOREGROUND/MAP DEMOFG
DIRECTORY *.OBJ
DELETE/NOQUERY GRAPH.*
DELETE/NOQUERY SUM.*
DELETE/NOQUERY MATCH.MAP
DEASSIGN

TIME
*EX (ESC) (ESC)
```

Close the file INDCT.COM.

EXECUTING AN INDIRECT FILE

Once you have created an indirect file with the editor and checked it for errors, you are ready to start its execution. You can run an indirect file under control of the single-job monitor or as the background job under control of the foreground/background system. While a foreground job is running however, you must take care to avoid conflicts between nondirectory-structured devices of the two jobs. For example, the jobs should not request the same magnetic tape or cassette.

The command to start the execution of an indirect file is the At sign (@) character 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:

```
.@INDCT (RET)
```

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.COM
.DATE 14-JAN-80
.TIME 8:00:00
.DATE
14-Jan-80
.DEASSIGN
.ASSIGN TT: LP:
.ASSIGN RK1: UOL:
```

```
.DIRECTORY/BRIEF VOL:
 14-Jan-80
GRAPH .FOR    MATCH .BAS    SUM   .MAC
 3 Files, 8 Blocks
 4754 Free blocks

.COPY VOL:GRAPH.FOR GRAPH.FOR

.COPY VOL:SUM.MAC SUM.MAC

.COPY VOL:MATCH.BAS MATCH.BAS

.FORTRAN/LIST GRAPH
FORTRAN IV      V02.1-10   Mon 14-Jan-80 08:00:13              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        LOGICAL*1 STRING(133),STAB(100)
0002        DATA XMIN,XMAX,MAXX/-5.0,5.0,45/
0003        DATA YMIN,YMAX,MAXY/-5.0,5.0,72/
0004        DATA FMIN,FMAX/0.0,1.0/
0005        SCAL(ZMIN,ZMAX,MAXZ,K)=ZMIN+FLOAT(K-1)*(ZMAX-ZMIN)/FLOAT(MAXZ-1)
0006        CALL SCOPY('- 1 2 3 4 5 6 7 8 9 +',STAB)
0007        MAXF=LEN(STAB)
0008        DO 20 IX=1,MAXX
0009        IIX=IX
0010        X=SCAL(XMIN,XMAX,MAXX,IIX)
0011        CALL REPEAT('*',STRING,MAXY)
0012        IF(IIX.EQ.1 .OR. IIX.EQ.MAXX) GOTO 20
0014        DO 10 IY=2,MAXY-1
0015        IIY=IY
0016        Y=SCAL(YMIN,YMAX,MAXY,IIY)
0017        IFUN=2+INT(FLOAT(MAXF-3)*(FUN(X,Y)-FMIN)/(FMAX-FMIN))
0018  10    STRING(IIY)=STAB(MINO(MAXF,MAXO(1,IFUN)))
0019  20    CALL PUTSTR(7,STRING,' ')
0020        CALL EXIT
0021        END
.MAIN.
FORTRAN IV      Storage Map for Program Unit .MAIN.

Local Variables, .PSECT $DATA, Size = 000474 (  158. words)

Name   Type  Offset     Name   Type  Offset     Name   Type  Offset
FMAX   R*4   000402     FMIN   R*4   000376     IFUN   I*2   000454
IIX    I*2   000436     IIY    I*2   000446     IX     I*2   000434
IY     I*2   000444     K      I*2   000430     MAXF   I*2   000432
MAXX   I*2   000362     MAXY   I*2   000374     MAXZ   I*2   000436
X      R*4   000440     XMAX   R*4   000356     XMIN   R*4   000352
Y      R*4   000450     YMAX   R*4   000370     YMIN   R*4   000364
ZMAX   R*4   000422     ZMIN   R*4   000416

Local and COMMON Arrays:

Name     Type   Section Offset  -------Size------ Dimensions
STAB   L*1      $DATA   000205  000144 (   50.) (100)
STRING L*1      $DATA   000000  000205 (   67.) (133)

Subroutines, Functions, Statement and Processor-Defined Functions:

Name   Type   Name   Type   Name   Type   Name   Type   Name   Type
EXIT    R*4   FLOAT   R*4   FUN     R*4   INT     I*2   LEN     I*2
MAXO    I*2   MINO    I*2   PUTSTR  R*4   REPEAT  R*4   SCAL    R*4
SCOPY   R*4
FORTRAN IV      V02.1-10   Mon 14-Jan-80 08:01:44              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)

Name   Type  Offset     Name   Type  Offset     Name   Type  Offset
FUN    R*4   000004 Eav 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
EXP     R*4   SQRT    R*4

.LINK/MAP GRAPH
RT-11 LINK  V06.01      Load Map        Mon 14-Jan-80 08:02:20
GRAPH .SAV      Title:  .MAIN.  Ident:  FORV02

Section  Addr   Size    Global  Value   Global  Value   Global  Value

. ABS.  000000 001000   (RW,I,GBL,ABS,OVR)
                        $USRSW  000000  $RF261  000000  $HRDWR  000002
                        $NLCHN  000006  $SYSV$  000011  $WASIZ  000131
                        $LRECL  000210  $TRACE  004737  $EAE    177304
 OTS$I   001000 014362   (RW,I,LCL,REL,CON)
                        $$OTSI  001000  $OTI    001026  $$OTI   001030
                        $$SET   002536  $CVTIF  003032  $CVTIC  003046
                        $CVTID  003046  CCI$    003060  CDI$    003060
                        $IC     003060  $ID     003060  CFI$    003074
                        $IR     003074  EXP     003160  SQRT    003520
                        MUF$PS  003714  MUF$MS  003720  MUF$IS  003732
                        $MULF   003740  MUF$SS  003752  $MLR    003752
                        DIF$PS  004350  DIF$MS  004354  DIF$IS  004366
                        $DIVF   004374  DIF$SS  004406  $DVR    004406
                        ADF$IS  005012  ADF$PS  005020  SUF$PS  005024
                        SUF$MS  005030  ADF$MS  005042  SUF$IS  005052
                        $ADDF   005060  $SUBF   005074  SUF$SS  005106
                        $SBR    005106  ADF$SS  005112  $ADR    005112
                        ADD$    005126  IDINT   005552  INT     005552
                        MAXO    005600  MINO    005624  CAI$    005650
                        CAL$    005656  ISN$    005706  $ISNTR  005712
                        LSN$    005726  $LSNTR  005732  MOI$IP  006066
                        MOI$SP  006070  MOI$PP  006076  MOI$MP  006102
                        MOI$PS  006112  MOI$PM  006120  MOI$PA  006126
                        MOI$OP  006134  MOI$IP  006142  MOF$SS  006152
                        MOF$SM  006164  MOF$SP  006174  LLE$    006200
                        LEQ$    006202  LGT$    006210  LGE$    006212
                        LNE$    006222  LLT$    006224  MOL$SM  006230
                        MOL$SA  006234  MOL$MS  006240  MOL$MM  006250
                        MOL$MA  006254  MOL$SP  006260  MOL$PP  006266
                        MOL$MP  006272  MOL$PM  006302  MOL$PS  006310
                        MOL$PA  006314  MOL$IM  006322  MOL$IA  006330
                        MOL$IP  006336  STK$L   006346  STK$I   006352
                        STK$F   006356  MOI$RS  006366  MOL$RS  006366
                        MOI$RM  006372  MOI$RP  006376  MOI$RA  006400
                        NGD$S   006404  NGF$S   006404  NGD$M   006416
                        NGF$M   006416  NGD$P   006432  NGF$P   006432
                        NGD$A   006436  NGF$A   006436  ADI$SS  006442
                        ADI$SA  006446  ADI$SM  006452  ADI$IS  006456
                        ADI$IA  006462  ADI$IM  006466  ADI$MS  006472
```

```
                        ADI$MA  006476  ADI$MM  006502  SUI$SS  006506
                        SUI$SA  006512  SUI$SM  006516  SUI$IS  006522
                        SUI$IA  006526  SUI$IM  006532  SUI$MS  006536
                        SUI$MA  006542  SUI$MM  006546  ICI$S   006552
                        ICI$M   006556  ICI$P   006562  ICI$A   006564
                        DCI$S   006570  DCI$M   006574  DCI$F   006600
                        DCI$A   006602  CMI$SS  006606  CMI$SI  006612
                        CMI$SM  006616  CMI$IS  006622  CMI$II  006626
                        CMI$IM  006632  CMI$MS  006636  CMI$MI  006642
                        CMI$MM  006646  NMI$IM  006652  NMI$II  006664
                        BLE$    006674  BEQ$    006676  BGT$    006704
                        BGE$    006706  BRA$    006710  BNE$    006714
                        BLT$    006716  MOF$RS  006726  MOF$RM  006734
                        MOF$RA  006744  MOF$RP  006750  MOF$MS  006754
                        MOF$PS  006766  MOF$MM  006772  MOF$MA  007004
                        MOF$MP  007012  MOF$PM  007020  MOF$PA  007024
                        MOF$PP  007030  IOR$    007034  AND$    007040
                        EQV$    007046  XOR$    007050  TSL$S   007064
                        TSL$M   007070  TSL$I   007074  TSL$P   007102
                        $OTIS   007110  $$OTIS  007112  RET$L   007232
                        RET$F   007236  RET$I   007244  RET$    007246
                        MOI$SS  007302  MOL$SS  007302  MOI$SM  007306
                        MOI$SA  007312  MOI$IS  007316  MOL$IS  007316
                        REL$    007316  MOI$IM  007322  MOI$IA  007326
                        MOI$MS  007332  MOI$MM  007336  MOI$MA  007342
                        MOI$OS  007346  MOI$OM  007352  MOI$OA  007356
                        MOI$1S  007362  MOI$1M  007370  MOI$1A  007376
                        EXIT    007404  SAL$IM  007410  SAL$SM  007412
                        SVL$IM  007420  SVL$SM  007422  SAL$MM  007430
                        SVL$MM  007434  $CVTFB  007440  $CVTFI  007440
                        $CVTCB  007454  $CVTCI  007454  $CVTDB  007454
                        $CVTDI  007454  CIC$    007466  CID$    007466
                        CLC$    007466  CLD$    007466  $DI     007466
                        CIF$    007476  CLF$    007476  $RI     007476
                        CIL$    007620  CLI$    007624  $ERRTB  007626
                        $SHORT  007626  $ERRS   007733  TVL$    010254
                        $TVL    010254  TVF$    010262  $TVF    010262
                        TVD$    010270  $TVD    010270  TVQ$    010276
                        $TVQ    010276  TVP$    010304  $TVP    010304
                        TVI$    010312  $TVI    010312  $STFS   010446
                        STF$    010454  $STF    010454  FOO$    010460
                        $EXIT   010500  END$    010624  ERR$    010636
                        $END    010650  $ERR    010666  IFW$    010710
                        $IFW    010714  IFW$$   010756  $CHKER  011026
                        $IOEXI  011052  $EOL    011100  EOL$    011102
                        $VRINT  011216  SAL$IP  011220  SAL$SP  011222
                        SVL$IP  011230  SVL$SP  011232  SAL$MP  011240
                        SVL$MP  011244  SAI$IM  011250  SAI$SM  011252
                        $BOUND  011256  SVI$IM  011302  SVI$SM  011304
                        SAI$MM  011314  SVI$MM  011320  SAVRG$  011324
                        THRD$   011502  $PUTRE  011504  $WAIT   012012
                        $FCHNL  012054  $INITI  012152  $CLOSE  012264
                        $PUTBL  012730  $GETBL  013140  $EOFIL  013324
                        $EOF2   013340  $FIO    014100  $$FIO   014104
                        $DUMPL  015234
OTS$P   015362 000050    (RW,D,GBL,REL,OVR)
SYS$I   015432 000244    (RW,I,LCL,REL,CON)
                        LEN     015432  REPEAT  015450  SCOPY   015602
USER$I  015676 000000    (RW,I,LCL,REL,CON)
$CODE   015676 001336    (RW,I,LCL,REL,CON)
                        $$OTSC  015676  FUN     016574  PUTSTR  016742
OTS$O   017234 001010    (RW,I,LCL,REL,CON)
                        $$OTSO  017234  $OPEN   017234
SYS$O   020244 000000    (RW,I,LCL,REL,CON)
$DATAP  020244 000106    (RW,D,LCL,REL,CON)
OTS$D   020352 000006    (RW,D,LCL,REL,CON)
OTS$S   020360 000002    (RW,D,LCL,REL,CON)
                        $AOTS   020360
SYS$S   020362 000004    (RW,D,LCL,REL,CON)
                        $SYSLB  020362  $LOCK   020364  $CRASH  020365
$DATA   020366 000542    (RW,D,LCL,REL,CON)
USER$D  021130 000000    (RW,D,LCL,REL,CON)
.$$$$.  021130 000000    (RW,D,GBL,REL,OVR)

Transfer address = 015676, High limit = 021126 = 4395.  words

.MACRO/LIST/CROSSREFERENCE SUM

SUM.MAC  VERSION 1      MACRO V04.00  14-JAN-80 08:07:25 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
     9                                  ;       'E' = THE SUM OF THE RECIPROCALS OF THE FACTORIALS
    10                                  ;       1/0! + 1/1! + 1/2! + 1/3! + 1/4! + 1/5! + ...
    11
    12 000000                           EXP:    .PRINT  #MESSAG          ;PRINT INTRODUCTORY TEXT
    13 000006  012705  000106                   MOV     #N,R5            ;NO. OF CHARS OF 'E' TO PRINT
    14 000012  012700  000107           FIRST:  MOV     #N+1,R0          ;NO. OF DIGITS OF ACCURACY
    15 000016  012701  000124'                  MOV     #A,R1            ;ADDRESS OF DIGIT VECTOR
    16 000022  006311                   SECOND: ASL     @R1              ;DO MULTIPLY BY 10 (DECIMAL)
    17 000024  011146                           MOV     @R1,-(SP)        ;SAVE #2
    18 000026  006311                           ASL     @R1              ;*4
    19 000030  006311                           ASL     @R1              ;*8
    20 000032  062621                           ADD     (SP)+,(R1)+      ;NOW *10, POINT TO NEXT DIGIT
    21 000034  005300                           DEC     R0               ;AT END OF DIGITS?
    22 000036  001371                           BNE     SECOND           ;BRANCH IF NOT
    23 000040  012700  000106                   MOV     #N,R0            ;GO THRU ALL PLACES, DIVIDING
    24 000044  014103                   THIRD:  MOV     -(R1),R3         ;BY THE PLACES INDEX
    25 000046  012702  177777                   MOV     #-1,R2           ;INIT QUOTIENT REGISTER
    26 000052  005202                   FOURTH: INC     R2               ;BUMP QUOTIENT
    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                                                                   ;FOR NEXT DIGIT
    32 000064  060261  177776                   ADD     R2,-2(R1)        ;GREATEST INTEGER CARRIES
    33                                                                   ;TO GIVE DIGIT
    34 000070  005300                           DEC     R0               ;AT END OF DIGIT VECTOR?
    35 000072  001364                           BNE     THIRD            ;BRANCH IF NOT
    36 000074  014100                           MOV     -(R1),R0         ;GET DIGIT TO OUTPUT
    37 000076  162700  000012           FIFTH:  SUB     #10.,R0          ;FIX THE 2.7 TO .7 SO
    38                                                                   ;THAT IT IS ONLY 1 DIGIT
    39 000102  103375                           BCC     FIFTH            ;(REALLY DIVIDE BY 10)
    40 000104  062700  000070                   ADD     #10+'0,R0        ;MAKE DIGIT ASCII
    41 000110                                   .TTYOUT                  ;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
    47 000124  000107                   A:      .REPT   N+1
    48                                          .WORD   1                ;INIT VECTOR TO ALL ONES
    49                                          .ENDR
    50
    51 000342     124     110     105  MESSAG: .ASCII  /THE VALUE OF E IS:/ <15><12> /2./ <200>
       000345     040     126     101
       000350     114     125     105
       000353     040     117     106
```

```
       000356     040     105     040
       000361     111     123     072
       000364     015     012     062
SUM.MAC  VERSION 1      MACRO V04.00  14-JAN-80 08:07:25 PAGE 1-1


       000367     056     200
    52                                 .EVEN
    53
    54          000000'                .END    EXP
SUM.MAC  VERSION 1      MACRO V04.00  14-JAN-80 08:07:25 PAGE 1-2
SYMBOL TABLE

A       000124R         FIFTH   000076R         FOURTH  000052R         N       = 000106          THIRD   000044R
EXP     000000R         FIRST   000012R         MESSAG  000342R         SECOND  000022R

. ABS.  000000      000
        000372      001
ERRORS DETECTED:  0

VIRTUAL MEMORY USED:  8448 WORDS  ( 33 PAGES)
DYNAMIC MEMORY AVAILABLE FOR  65 PAGES
DK:SUM,LP:SUM=DK:SUM/C

ERRORS DETECTED:  0

SUM.MAC  VERSION 1      MACRO V04.00  14-JAN-80 08:07:25 PAGE S-1
CROSS REFERENCE TABLE (CREF V04.00 )


A       1-15    1-47#
EXP     1-12#   1-54
FIFTH   1-37#   1-39
FIRST   1-14#   1-44
FOURTH  1-26#   1-28
MESSAG  1-12    1-51#
N       1-7#    1-13    1-14    1-23    1-47
SECOND  1-16#   1-22
THIRD   1-24#   1-35
SUM.MAC  VERSION 1      MACRO V04.00  14-JAN-80 08:07:25 PAGE M-1
CROSS REFERENCE TABLE (CREF V04.00 )


.EXIT   1-3#    1-45
.PRINT  1-3#    1-12
.TTYOU  1-3#    1-41

.LINK/MAP SUM
RT-11 LINK  V06.01      Load Map        Mon 14-Jan-80 08:10:57
SUM   .SAV      Title:  SUM.MA  Ident:

Section  Addr   Size    Global  Value   Global  Value   Global  Value

 . ABS.  000000 001000   (RW,I,GBL,ABS,OVR)
         001000 000372   (RW,I,LCL,REL,CON)

Transfer address = 001000, High limit = 001370 = 380.  words

.RENAME MATCH.BAS MATCH.MAP

.MACRO/LIST/CROSSREFERENCE DEMOFG

DEMOFG  MACRO V04.00  14-JAN-80 08:11:15 PAGE 1


     1                                          .TITLE  DEMOFG
     2                                          .IDENT  /V03.01/
     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,.MRKT,.QSET,.SPND
     7
     8 000000                           START:: .PRINT  #MSG                    ;PRINT INTRODUCTORY MESSAGE
     9 000006                                   .QSET   #QUEUE,#100.            ;SET ASIDE 100 0 ELEMENTS FOR MESSAGES
    10 000020                                   .MRKT   #AREA,#TIME,#MKTC,#1    ;SET UP MKTIM FOR 2 SECONDS FROM NOW
    11 000054                                   .SPND                           ;SUSPEND THE FG TILL MKTIM SATISFIED
    12
    13                                  ;       MKTIM COMPLETION ROUTINE
    14
    15 000062  026727  000314  000132  MKTC:   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  005267  000250                   INC     MSGCNT                  ;BUMP MESSAGE COUNTER
    19 000132                           MKTC1:  .MRKT   #AREA,#TIME,#MKTC,#1    ;SET UP ANOTHER MKTIM FOR 2 SECONDS
    20 000166  000207                           RETURN                          ;RETURN FROM COMPLETION ROUTINE
    21
    22                                  ;       SDAT COMPLETION ROUTINE
    23
    24 000170  005367  000206          SDATC:  DEC     MSGCNT                  ;ONE OF THE MESSAGES HAS BEEN RECEIVED
    25 000174  000207                           RETURN                          ;RETURN(RTS PC)
    26
    27                                  ;       ASCII MESSAGES
    28                                          .NLIST BEX
    29
    30 000176     106     117     122  MSG:    .ASCII  /FOREGROUND DEMONSTRATION PROGRAM/<15><12>
    31 000240     123     105     114          .ASCII  /SENDS A MESSAGE TO THE BACKGROUND PROGRAM "DEMOBG"/<15><12>
    32 000324     105     126     105          .ASCIZ  /EVERY 2 SECONDS, TELLING IT TO RING THE BELL./
    33                                          .EVEN
    34
    35 000402  000000                  MSGCNT: .WORD   0                       ;MESSAGE COUNTER
    36
    37                                                                          ;TIME CONSTANT
    38 000404  000000                  TIME:   .WORD   0                       ;HIGH ORDER
    39 000406  000170                          .WORD   60.*2                   ;60 TICKS A SECOND,2 SECONDS
    40
    41 000410                          AREA:   .BLKW   6                       ;EMT ARGUMENT AREA
    42
    43 000424                          BUFFER: .BLKW   400                     ;BUFFER FOR MESSAGES
    44
    45                                                                          ;QUEUE AREA
    46 001424                          QUEUE:  .BLKW   100.*10.                ;10. WORDS PER QUEUE ELEMENT FOR THE XM MONITOR
    47
    48          000000                         .END    START
DEMOFG  MACRO V04.00  14-JAN-80 08:11:15 PAGE 1-1
SYMBOL TABLE

AREA    000410R         MKTC1   000134R         QUEUE   001424R         SDATC   000170R         TIME    000404R
BUFFER  000424R         MSG     000176R         START   000000R         .MRKT   000000RG        .QSET   000020RG
MKTC    000062R         MSGCNT  000402R

. ABS.  000000      000
        005344      001
ERRORS DETECTED:  0

VIRTUAL MEMORY USED:  9472 WORDS  ( 37 PAGES)
DYNAMIC MEMORY AVAILABLE FOR  65 PAGES
DK:DEMOFG,LP:DEMOFG=DK:DEMOFG/C

ERRORS DETECTED:  0

DEMOFG  MACRO V04.00  14-JAN-80 08:11:15 PAGE S-1
CROSS REFERENCE TABLE (CREF V04.00 )
```

```
...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     1-41#
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     1-18*    1-24*    1-35#
QUEUE    1-9      1-46#
SDATC    1-17     1-24#
START    1-8#     1-48
TIME     1-10     1-19     1-38#
DEMOFG  MACRO V04.00  14-JAN-80 08:11:15 PAGE M-1
CROSS REFERENCE TABLE (CREF V04.00 )


...CM0   1-9
...CM1   1-17
...CM2   1-10     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
.MRKT    1-6#     1-10     1-19
.PRINT   1-6#     1-8
.QSET    1-6#     1-9
.SDATC   1-6#     1-17
.SPND    1-6#     1-11

.LINK/FOREGROUND/MAP DEMOFG
RT-11 LINK  V06.01      Load Map        Mon 14-Jan-80 08:14:30
DEMOFG.REL      Title:  DEMOFG  Ident:  V03.01

Section  Addr   Size    Global  Value   Global  Value   Global  Value

 . ABS.  000000 001000   (RW,I,GBL,ABS,OVR)
         001000 005344   (RW,I,LCL,REL,CON)
                       START   001000

Transfer address = 001000, High limit = 006342 = 1649.  words

.DIRECTORY *.OBJ
 14-Jan-80
ODT     .OBJ      9  19-Nov-79      VDT     .OBJ      9  19-Nov-79
VTHDLR.OBJ      9  19-Nov-79      PLOT55.OBJ      3  19-Nov-79
LIBSYS.OBJ     47  19-Nov-79      GRAPH .OBJ     16  14-Jan-80
SUM     .OBJ      1  14-Jan-80      SYSLIB.OBJ    245  25-Nov-79
DEMOFG.OBJ      1  14-Jan-80
 9 Files, 340 Blocks
 1064 Free blocks

.DELETE/NOQUERY GRAPH.*

.DELETE/NOQUERY SUM.*

.DELETE/NOQUERY MATCH.MAP

.DEASSIGN

.TIME
08:15:03

.
```

SUMMARY: COMMAND 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

This indirect file 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-5279B-TC). Maynard, Mass.: Digital Equipment Corporation, 1980.

A guide to the use of the RT-11 operating system. See Chapter 4.

CHAPTER 17
ADVICE TO NEW USERS

This manual introduces you to several common RT-11 functions but is neither exhaustive nor comprehensive in its treatment of system features, commands, or their options. For many, these fundamental system operations are sufficient; other users, however, may need or want to learn 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 *RT-11 Documentation Directory,* or the *RT-11 System User's Guide.* The *RT-11 Documentation Directory* lists all RT-11-related material available from DIGITAL: the User's Guide explains in detail each command contained in this manual and additional monitor commands, including all possible command options.

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, do not insert text in large segments. Divide long editing sessions into short ones so that user (or hardware) errors do not cost 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 non-directory-structured devices (LP:, MT:, CT:, PC:, TT:) used by the two jobs.

USING THE HELP FILE

HELP

A HELP file is distributed with RT-11 that contains information about the keyboard monitor commands and how to use them. 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

```
.HELP (RET)
```

The following information is provided:

```
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 topac 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
```

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 (:).

There are only two options you can use with the HELP command: they 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 output to the terminal.

To get all the information in the help file about the keyboard monitor command ASSIGN, type

```
.HELP ASSIGN RET
```

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.

```
.HELP DIRECTORY OPTIONS (RET)
```

lists all the options that are available for use with the DIRECTORY keyboard monitor command.

If you need information only about a specific item in a list of options, type the item in the command line as follows:

```
.HELP DIRECTORY OPTIONS:ORDER (RET)

 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

This appendix describes the manual bootstrapping procedures used for PDP-11 computers that do not have the automatic bootstrapping capability described in Chapter 2. Three categories are covered:

Typing the Bootstrap on the Terminal Keyboard
Using a Pushbutton Console to Bootstrap
Using a Switch Register Console to Bootstrap

TYPING THE BOOTSTRAP ON THE TERMINAL KEYBOARD

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 and System Generation Guide*, and copy the numbers into the space below:

Now, type each number in the column on your terminal keyboard using the following method (if you make a mistake, type the DELETE key on the terminal keyboard, once for each typing error, and then retype the digit[s]):

1. Type 001000.
2. Type slash (/).
3. Type the first number in the bootstrap column.
4. Type the LINE FEED key on the keyboard.
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 on the keyboard.
8. Type 1000G.
9. Continue to step 11 in Chapter 2.

USING A PUSHBUTTON CONSOLE TO BOOTSTRAP

If your computer has a pushbutton console on its front panel similar to that 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 and System Generation Guide*, and copy the numbers into the space provided below. If your system has a hardware bootstrap,[1] 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, set the numbers into the pushbuttons using the following method (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 Chapter 2.

[1] 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, set the numbers into the pushbuttons, using the following method (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 Chapter 2.

USING A SWITCH REGISTER CONSOLE TO BOOTSTRAP

If your computer has a switch register console on the front panel similar to those shown in Figure A–2, you can use the switches to manually give the computer the bootstrapping information it needs to start the system.

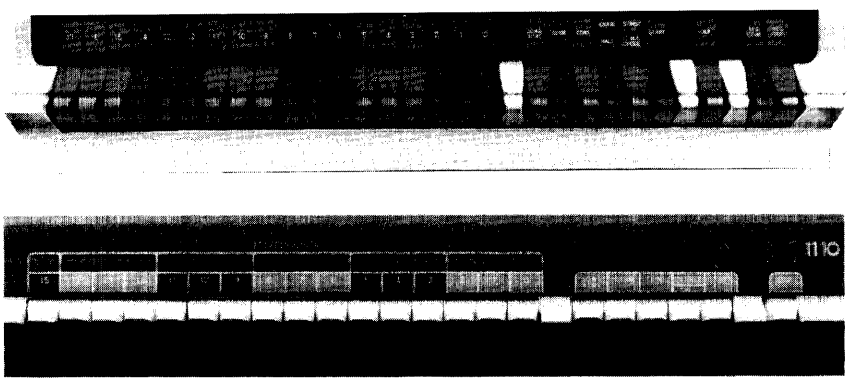


Figure A–2 Switch Register Consoles

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.

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 switch register console. First, obtain the bootstrap of your system device from the *RT-11 Installation and System Generation Guide*, and copy the numbers into the space provided below. If your system has a hardware bootstrap,[1] the bootstrap consists of only two numbers, which you should copy into the left-hand space; otherwise, the bootstrap consists of two columns of numbers labeled Location and Contents, which you should copy into the right-hand space:

Hardware Bootstrap | Other Bootstraps

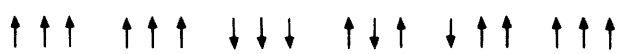
Load Address =
Start Address =

Next, convert the numbers in the column to binary numbers, using the conversion process shown in Table A-1.

Table A-1: Binary Conversion

Octal		Binary
0	=	000
1	=	001
2	=	010
3	=	011
4	=	100
5	=	101
6	=	110
7	=	111

For example, the number 173100 is converted to 001 111 011 001 000 000. You set this 18-digit binary number into the switch register by placing each individual switch in an up position for a 1 or a down position for a 0. The number 173100 is set into the switch register as follows:



The number 012700 is converted to 000 001 010 111 000 000 and is set into the switch register as follows:

[1] 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.

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 Chapter 2.

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 Chapter 2.

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.

STOPPING AND STARTING THE SYSTEM

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 the System

Perform the following steps in order:

1. Stop the computer. Press the HALT switch if your computer operator's console has switches; hold the CNTRL button down and push the HLT/SS button if your computer operator's console has pushbuttons.

2. Unload the system volume. Turn the device unit to an off-line condition, and remove the system volume.

3. Unload the storage volume. Turn the device unit to an off-line condition, and remove the storage volume.

4. Remove and save all terminal and line printer output listings.

Starting the System

Perform the following steps in order:

1. Follow the bootstrap procedure in Chapter 2.

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. Once the problem is diagnosed, start 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 locate the bootstrapping information provided by the DIGITAL representative who installed your system.

   First, if an experienced RT-11 user is available to help you, ask this person to fill in the missing information in the *RT-11 Installation and System Generation Guide.* Then retry the bootstrap procedures in Chapter 2 of this manual.

   If no one is available to help you, 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. Read the system build and start operations that are outlined in the *RT-11 Installation and System Generation Guide.* Then try the bootstrap procedures in Chapter 2 again.

2. You have followed the bootstrapping instructions correctly, but your system printed a message other than what you expected.

   a. If the message is one of the following:

   ```
   ?BOOT-U-Insufficient memory

   ?BOOT-U-I/O error

   ?BOOT-U-No memory management hardware

   ?BOOT-U-Monitor file on volume
   ```

   it is a bootstrap error message, indicating that a problem in the system is preventing bootstrapping. These four messages are fully explained in the *RT-11 System Message Manual,* but you should not attempt to correct the problem yourself if an experienced user is available to help.

   b. If the message is one of the following:

   ```
   RT-11FB     V04.xx

   RT-11XM     V04.xx
   ```

a valid RT-11 V4 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 (sy is the appropriate two-character code for your system volume — see question 5 in the Hardware Configuration section of Chapter 2); (RET) indicates that you should type the RETURN key on your terminal keyboard:

```
.BOOT (RET)
Device or file? RT11SJ.SYS (RET)
```

c. Any other message indicates that an old version of RT-11 (V1, V2, V2B, V2C) has been bootstrapped. Only Version 3 and later releases 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.

   Retry the bootstrap procedure from the beginning. Before you begin, be sure that the system volume is properly mounted in device unit 0. Check that the computer is on but is not running (the light labeled RUN should not be lit); if it is running, stop it as described above. Check that the terminal is on line and that its baud rate switch (if present) is set to 300. If you are using a display, be sure the screen is bright enough. If your terminal uses a paper printer, be sure that the paper is properly loaded.

BACKING UP THE SYSTEM VOLUME

A copy of the system volume should have been made during system installation. This copy is called the master copy and should be stored for safekeeping. If you cannot locate a master copy for your system volume, make one before you continue. Backup instructions are in the *RT-11 Installation and System Generation Guide* and should be performed by an experienced user.

DIRECTORY VS NONDIRECTORY-STRUCTURED VOLUMES

Storage volumes are called file-structured volumes because they are capable of physically storing files. They can be further categorized as directory-structured and nondirectory-structured volumes based on their method of directory information storage, collection, and printing.

The directory information kept on a volume includes file names and file types, dates of creation, and (in most cases) file lengths. When you type the DIRECTORY command, this directory information prints on the terminal. Volumes such as disk, diskette, and DECtape keep this information in a single place at the beginning of the volume. Each time you add or erase a file, the directory information at the beginning of the volume is updated accordingly. Thus, these volumes have a true volume directory and are said to be directory-structured. Magtape and cassette volumes, on the other hand, do not keep directory information in any single place on the tape but rather with each individual file. Their directory information is obtained by sequentially reading through all the files on the tape and collecting the directory for printing as each file is encountered. Thus, these volumes are said to be nondirectory-structured.

You can list the volume directories in either a complete or an abbreviated format. Complete volume directories include the file name, file type, file length (usually), and date of creation if you entered a date via the DATE command before creation. For most volumes, the directory format is as follows:

```
16-Jan-80
 FILE .TYP    26  23-JUN-77
```

Cassette directories are slightly different. Their directories do not indicate file lengths but instead show a sequence number for each file:

```
16-Jan-80
 FILE .TYP    0  23-JUN-77
```

The sequence number simply indicates whether the file is continued from another cassette. The number 0 means the file is not continued from another cassette while any other number indicates that the file is continued. The number of blocks printed at the end of a cassette directory does not represent the total size of the files on the volume but instead represents the total of the sequence numbers.

Abbreviated volume directories are handled in the same way for all directory-structured volumes; they include only the file name and file type, and are printed in five columns on the terminal. For more information about directory-structured and nondirectory-structured volumes, see the *RT-11 System User's Guide*, Chapter 3.

ALTERNATE RENAME OPERATION FOR MAGTAPE AND CASSETTE USERS

Because of the sequential (nondirectory-structured) nature of magtapes and cassettes, you cannot use the RENAME monitor command. To perform the RENAME operation, you must first copy the file, using the new file name, and then erase the old file name.

Thus, to change the name of GRAPH.TWO on your magtape or cassette 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)
```

Now there are two copies of the GRAPH file. Erase 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/NOQUERY (RET)
Files? VOL:GRAPH.TWO (RET)
```

Short Command Format

```
.DELETE/NOQUERY VOL:GRAPH.TWO (RET)
```

A single copy of GRAPH.FOR now resides on your default storage (system) volume. Copy the file onto your MT: or CT: 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/NOQUERY (RET)
Files? GRAPH.FOR (RET)
```

Short Command Format

```
.DELETE/NOQUERY GRAPH.FOR (RET)
```

The combined effect of these four commands is to "rename" GRAPH.TWO to GRAPH.FOR.

USING THE FORTRAN/BASIC LANGUAGE VOLUME

During system installation, a special system volume was created specifically for your use with this manual. This volume contains the FORTRAN and/or BASIC language processors and those monitor files required to use these language processors. Before you can perform the FORTRAN or BASIC demonstrations, you must substitute this 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 and BASIC demonstrations.

Make sure no system operations are in progress (the monitor prompting 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 the language volume, write-protected. Bootstrap the system (see Stopping and Starting the System, this appendix). The following monitor message should appear:

```
RT-11SJ     V04.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 the manual.

SUBSTITUTING VOLUMES DURING OPERATIONS

Diskette users and FORTRAN users who have the FORTRAN language processor on a volume apart from their system volume must occasionally perform the kinds of file copying and volume swapping operations that follow. These operations are necessary when the files you need to use are not stored on the volume(s) currently mounted. The situation requires that you make the appropriate volume substitutions before you continue.

Thus, before you can compile the FORTRAN file THIRD.FOR, you must substitute the language volume containing the FORTRAN 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)
```

Stop the system, remove the system volume currently loaded in unit 0, and insert the language volume write-protected. See Stopping and Starting the System (this appendix) if necesary. The following message should appear when you bootstrap the language volume.

```
RT-11SJ     V04.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.

Next, compile the FORTRAN program THIRD.FOR, which is now on VOL:

Long Command Format

```
.FORTRAN (RET)
Files? VOL:THIRD.FOR (RET)
PUTSTR
```

Short Command Format

```
.FORTRAN VOL:THIRD (RET)
PUTSTR
```

This command causes the object module to be created on the default storage volume (DK:), which is presently the system volume (that is, the language volume). If errors occur during the compile operation, they indicate that you have incorrectly typed the source file. In this case, you must edit the file THIRD.FOR, recompile, and then copy the file to VOL:. Once you have an object module that compiles without error and is stored on VOL:, reload the main system volume in unit 0. Again, follow the directions in Stopping and Starting the System. Once you have

bootstrapped the volume, 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 (RET)
From? VOL:THIRD OBJ (RET)
To  ? THIRD.OBJ (RET)
```

Short Command Format

```
.COPY VOL:THIRD.OBJ THIRD.OBJ (RET)
```

Return to Chapter 13, to the section entitled Building the Object Library.

USING THE LINK VOLUME

During system installation, a special system volume was created 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 (RET)
From? GRAPH.OBJ (RET)
To  ? VOL:GRAPH.OBJ (RET)
```

Short Command Format

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

Now, make sure no system operations are in progress (the monitor prompting period should appear at the left margin of the terminal printer), and stop the system (see Stopping and Starting the System, this appendix). Next, remove the system volume currently loaded in device unit 0, and insert the LINK volume, write-protected if possible. Bootstrap the system (see Stopping and Starting the System, this appendix). The following monitor message should appear:

```
RT-11SJ     V04.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 (RET)
From? VOL:GRAPH.OBJ (RET)
To  ? GRAPH.OBJ (RET)
```

Short Command Format

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

When you have done this, you are ready to run the linking demonstration. Return to the main text of the manual.

FORTRAN/LINK FILE MAINTENANCE

Follow the file maintenance operations outlined in this section if you substituted both a FORTRAN language volume and a LINK volume to perform the demonstrations in Chapter 9.

First, mount the FORTRAN 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 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)
 16-Jan-80
GRAPH .BAK      2  19-Nov-79       GRAPH .OBJ     14  08-Jan-80
GRAPH .FOR      2  08-Jan-80       GRAPH .LST     10  08-Jan-80
 4 Files, 28 Blocks
 48 Free blocks
```

The fact that you have corrected errors in the source file GRAPH.FOR makes the version on your storage volume obsolete. Thus, 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)
```

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 files of value to your storage volume, delete the useless files from the system volume:

Long Command Format

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

Short Command Format

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

Make sure no system operations are in progress (the monitor prompting period should appear at the left margin of the terminal printer), and stop the system (see Stopping and Starting the System, this appendix). Next, remove the system volume currently loaded in device unit 0, and insert the LINK volume, write-protected if possible. Bootstrap the system (see Stopping and

Starting the System, this appendix). The following monitor message should appear:

```
RT-11SJ     V04.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
 16-Jan-80
GRAPH .OBJ    14  08-Jan-80      GRAPH .SAV    19  08-Jan-80
 2 Files, 33 Blocks
 80 Free blocks
```

Transfer GRAPH.SAV to your storage volume. This allows you to rerun 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 useless files from your system volume:

Long Command Format

```
.DELETE RET
Files? GRAPH.OBJ,GRAPH.SAV RET
 Files deleted:
DK.GRAPH.OBJ   ? Y
DK.GRAPH.SAV   ? Y
```

Short Command Format

```
.DELETE GRAPH.OBJ,GRAPH.SAV (RET)
 Files deleted:
DK.GRAPH.OBJ  ? Y
DK.GRAPH.SAV  ? Y
```

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

```
.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 required 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 (voltages) 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

Referring to the subset of ASCII characters that includes the 26 alphabetic characters and the 10 numeric characters.

ANSI

American National Standards Institute.

APL (A Programming Language)

A condensed, high-level language capable of describing complex information processing in convenient notation. It uses arrays as basic data elements and manipulates them with a set of powerful operators. Statements are usually interpreted during execution and require no compilation whatsoever.

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 its 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 using a coded character set 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 by replacing symbolic operation codes with binary operation codes, and symbolic addresses with absolute or relocatable addresses.

Assembly language

A symbolic programming language that normally can be translated directly into machine language instructions and is, therefore, specific to a given computing system.

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

Pertaining to an event triggered by the occurrence of an unrelated event rather than "synchronous" or related operations scheduled by time intervals.

Background program

A program operating automatically, at a low priority, when a higher priority (foreground) program is not using system resources.

Backup file

A copy of a file created in case the primary file is lost or destroyed.

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 (Beginner's All-purpose Symbolic Instruction Code)

An interactive, "algebraic" type of 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 signaling speed.

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 specified size that is peculiar to a device. The smallest system-addressable segment on a mass storage device in reference to I/O.

Bootstrap

A technique or routine whose first instructions are sufficient to load the remainder of itself and start a complex system of programs.

BOT (Beginning Of Tape)

A reflective marker applied to the backside of magtape, which 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 that may cause erroneous results.

Bus

A flat, flexible cable that consists 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 unit of a computer that includes the circuits controlling 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. (The character pointer is visible as a blinking cursor on VT-11 display hardware.) 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 erase the contents of a storage location by replacing the contents, normally with 0s or spaces.

Clock

A device that generates regular periodic signals for synchronization.

Code

A system of symbols and rules used for representing information — usually refers to instructions executed by computer.

Coding

The writing of instructions for a computer, using symbols meaningful to the computer itself or to an assembler, compiler, or other language processor.

Command

A word, mnemonic, or character that, by virtue of its syntax in a line of input, 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 to a computer system that generally includes a command, one or more file specifications, and optional qualifiers.

Compile

To produce binary code from symbolic instructions written in a high-level source language.

Compiler

A program that translates a high-level source language into a language suitable for a particular machine.

Computer

A machine that can be programmed to execute a repertoire of instructions. Programs must be stored in the machine before they can be executed.

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 strings of characters to produce a longer 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 particular 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. It is used to initiate and direct overall system operation through software running 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, as in background/foreground programming.

Conversational

See Interactive.

CPU

See Central processing unit.

Crash

A hardware crash is the complete failure of a particular device, sometimes affecting the operation of an entire computer system. A software crash is the complete failure of an operating system, usually characterized by some failure in the system's protection mechanisms or flaw in the executing software.

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 symbols used in a source program and the statements where they 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 any or all facts, numbers, letters, and symbols. Basic elements of information that can be processed by a computer.

Data base

An organized collection of interrelated data items that allows one or more applications to process the items without regard to physical storage locations.

Data collection

The act of bringing data from one or more points to a central point 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, card reader, etc. Often used erroneously to mean "volume."

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 drives or services an I/O device and controls the physical hardware activities on the 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 usually consists of a two-character device mnemonic followed by an optional device unit number and a colon. For example, the common device name for RK05 disk drive unit 1 is "RK1:."

Device unit

One of a set of similar peripheral devices (for example, disk unit 0, DECtape unit 1, etc.). May be used synonymously with volume.

Diagnostics

A set of procedures for the detection and isolation of a malfunction or mistake.

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 table that contains the names of and pointers to files on a mass storage volume.

Directory-structured

Refers to a storage volume with a true volume directory at its beginning that 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 portray 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 by an I/O device, such as terminal or CRT, of characters typed by the programmer.

Edit

To arrange and/or modify the format of data (for example, to insert or delete characters).

Editor

A program that interacts with 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 actually 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 true, specified, or theoretically correct value or condition.

Execute

To carry out 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

Historically used synonym 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 treated as a unit, which occupies one or more blocks on a mass storage volume such as disk or magtape, and has an associated file name (and file 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 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 identifying the volume on which the file is stored, 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 noting 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 and another part represents the significant digits or fractional part (for example, 5.39×10^{8} — Decimal; 137.3×8^{4} — Octal; 101.10×2^{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 (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.

Garbage

Meaningless signals or bit patterns in memory.

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 the module in which they are defined and as externals in the other modules that use them.

Hack

A seemingly inspired, but obscure, solution that is superior by some measure to a straightforward one.

Half duplex

Pertaining to a communication system in which two-way communication is possible, but only one way at a time.

Handler

See Device handler.

Hardware

The physical equipment components of a computer system.

Hardware bootstrap

A bootstrap that is inherent in the hardware and need only be activated by specifying the appropriate load and start address.

High-level language

A programming language whose statements are typically translated into more than one machine language instruction. Examples are BASIC and FORTRAN.

High-order byte

The most significant byte in a word. The high-order byte occupies bit positions 8 through 15 of a PDP–11 word and is always an odd address.

Image mode

A mode of data transfer, in which each byte of data is transferred without any interpretation or data changes.

Indirect address

An address that specifies a storage location containing either a direct (effective) address or another indirect (pointer) address.

Indirect file

A file containing commands that are processed sequentially, but that could have been entered interactively at a terminal.

Industry-standard

A condition, format, or definition that is accepted as the norm by the majority of the (computer) industry.

Initialize

To set counters, switches, or addresses to starting values at prescribed points in the execution of a program, particularly in preparation for re-execution of a sequence of code. To format a volume in a particular file-structured format in preparation for use by an operating system.

Input

The data to be processed; the process of transferring data from external storage to internal storage.

Input/Output device

A device attached to a computer that makes it possible to bring information into the computer or get information out.

Instruction

A coded command that tells the computer what to do and where to find the values it is to work with. A symbolic instruction looks like ordinary language. Symbolic instructions must, however, be changed into machine instructions (usually by another program) before they can be executed by the computer.

Interactive processing

A technique of user/system communication in which the operating system immediately acknowledges and acts upon requests entered by the user at a terminal. Compare with batch processing.

Interface

A shared boundary. An interface might be a hardware component to link two devices, or it might be a portion of storage or registers accessed by two or more computer programs.

Internal storage

The storage facilities forming an integral physical part of the computer and directly controlled by the computer, for example, the registers of the machine and main memory.

Interpreter

A computer program that translates, then executes, a source language statement before translating (and executing) the next statement.

Interrupt

A signal that, when activated, causes a transfer of control to a specific location in memory, thereby breaking the normal flow of control of the routine being executed.

Interrupt-driven

Pertaining to software that uses the interrupt facility of a computer to handle I/O and respond to user requests: RT-11 is such a system.

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, for example, a program and all of its related subroutines, data, and control statements; also, a batch control file.

Label

One or more characters used to identify a source language statement or line.

Language

A set of representations, conventions, and rules used to convey information.

Latency

The time from 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 CRT screen. Used to input information to a CRT display system.

Linkage

In programming, 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 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 are not presently (and may never be) candidates for swapping or transferring around.

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 that can be 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 actual 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 command 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. Normally, the main program exercises primary control over the operations performed and calls subroutines or subprograms to perform specific functions.

Manual input

The entry of data by hand into a device at the time of processing.

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 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. In the strict sense, memory 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, etc.

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, 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 paper tape, 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 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. The user communicates in octal notation.

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 or executive 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 from among several alternatives associated with the command. In the RT-11 computer 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 program 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 50–60 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.

Patch

To modify a routine in a rough or expedient way, usually by modifying the binary code rather than by re-assembling it.

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 undergoing execution and manipulation by a computer.

Processor

In hardware, a data processor. In software, a computer program that includes the compiling, assembling, translating, and related functions for a specific programming language (for example, FORTRAN 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 locations in memory (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 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)

See Random access.

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

Computation performed while a related or controlled physical activity is occurring so that the results of the computation can be used in 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 sharable segment of pure code and a nonsharable 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 normally 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 (for example, compilers, library routines).

Software bootstrap

A bootstrap that is activated by manually 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, easily understood by people, 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 even 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, rather than processing data to produce results.

Swapping

The process of moving data from memory to a mass storage device, temporarily using the evacuated 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. Any program that is part of or supplied with the basic operating system (for example, a system utility 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 LA36 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 person.

Time sharing

A method of allocating resources to multiple users so that the computer, in effect, 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 program 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 treated as file-structured data storage.

Wildcard

A representation of file names and file types in a file specification with asterisks or partially with question marks.

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 transfers that would write information on it.

Write-protected

The condition of a volume that is protected against transfers that would write information on it.

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

P

Q

R

S

T

U

V

W

READER'S COMMENTS

NOTE: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. If you require a written reply and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

Did you find errors in this manual? If so, specify the error and the page number.

Please indicate the type of user/reader that you most nearly represent.

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Other (please specify)_____

Name_____ Date_____

Organization_____

Street_____

City_____ State_____ Zip Code_____
or
Country

Do Not Tear - Fold Here and Tape

digital

No Postage Necessary if Mailed in the United States

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO.33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

BSSG PUBLICATIONS TW/A14
DIGITAL EQUIPMENT CORPORATION
1925 ANDOVER STREET
TEWKSBURY, MASSACHUSETTS 01876

Do Not Tear - Fold Here

READER'S COMMENTS

NOTE: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. If you require a written reply and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

Did you find errors in this manual? If so, specify the error and the page number.

Please indicate the type of user/reader that you most nearly represent.

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Other (please specify)_____

Name_____ Date_____

Organization_____

Street_____

City_____ State_____ Zip Code_____
or
Country

— — — Do Not Tear - Fold Here and Tape —

digital

No Postage Necessary if Mailed in the United States

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO.33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

BSSG PUBLICATIONS TW/A14
DIGITAL EQUIPMENT CORPORATION
1925 ANDOVER STREET
TEWKSBURY, MASSACHUSETTS 01876

— — Do Not Tear - Fold Here —