Working with RT-11

David Beaumont
Anne Summerfield
Julie Wright

The RT-11 Technical User's Series

Working with RT–11

David Beamont
Anne Summerfield
Julie Wright

digital
DECbooks

Designed by Virginia J. Mason
Printed in the United States of America

10 9 8 7 6 5 4 3 2

Documentation number EY-00021-DP
ISBN 0-932376-31-2

DEC	PDP	RSX
DECtape	PDT	RT–11
DIBOL	Professional	VT
	RSTS	

Library of Congress Cataloging in Publication Data

Beaumont, David, 1938–
Working with RT–11.

(The RT–11 technical user's series)
Bibliography: p.
Includes index.
1. RT–11 (Computer operating system) I. Summerfield, Anne, 1958– II. Wright, Julie, 1955–
III. Title. IV. Series.
QA76.7.B424 1984 001.64'2 83-26185
ISBN 0-932376-31-2

Contents

Acknowledgment

We would like to thank all those who contributed to the material in this publication. In particular, we are grateful to Dan McCarthy, who donated his time to review the manuscript and revise practice exercises; Les Parent, who took time from his busy schedule to review the manuscript; Marcy Samel and Judith Scott, who informed us of the latest developments in the RT–11 operating system. The staff at Digital Press deserve special commendation for their invaluable assistance.

Introduction

Working with RT–11 introduces you to the components and functions of RT–11—a single-user, real-time operating system that can handle multiple tasks. This book assumes that you have previous experience working at a computer terminal with an operating system other than RT–11, that you understand how a computer works, and that you are familiar with terms such as bit, byte, word, and file.

The first three chapters of *Working with RT–11* deal with system organization. Chapter 1, "Identifying RT–11 Components," examines the hardware requirements and software components of an RT–11 system. Particular attention is paid to major monitor components. Chapter 2, "Getting Started," puts you at the terminal to begin working with the keyboard, getting HELP, and using some essential control characters. Chapter 3, "Storing Data on Disks," describes how file storage is organized, how files are specified, and how information in files can be manipulated using simple commands.

Two chapters examine the editors most often used on RT–11 systems. Chapter 4, "Using KED to Edit Text Files," explains how to use a screen editor to create and edit text files. Chapter 5, "Editing and Issuing Monitor Commands," discusses the single line editor used to modify command lines, and describes how to issue commands using Digital Command Language (DCL), Command String Interpreter (CSI), and Concise Command Language (CCL).

The flexibility of the RT–11 system for programming and using prepared applications is pointed up in the next two chapters. Chapter 6, "Using Utility Programs," describes the functions of utility programs and shows how to call up utilities with DCL commands. Chapter 7, "Developing Programs," tells you how to use the RT–11 system to create, compile, and run a program.

The last two chapters in the book focus on ways to save you time and make the most efficient use of space in your files. Chapter 8, "Creating Files of Commands," details the creation and function of indirect command files and indirect control files. Chapter 9, "Conserving Space with Device Support," tells you how to make use of virtual memory and how to create and mount logical disks.

When you have read the chapters and completed the practices, you will be familiar with the organization of the RT–11 system. You will be confident in using DCL commands to operate on files and in accessing the system utilities to edit and run your own programs. And you will be able to use indirect files to process sequences of monitor commands.

Learning Approach

After you have read the introduction and the chapter text, complete any practices. Many practices are not designed to have right or wrong answers but to give you experience working with your RT–11 system. Answers to practices that require them are provided near the end of the chapter. If you need further study before moving on to the next chapter, read the materials suggested at the end of the chapter, then review the chapter from the beginning.

To become familiar with RT–11, you should spend at least half a day at a time reading and working with the system and not allow more than a week between work sessions. You should try to finish the material covered in this book within two weeks.

If possible, you should work in a quiet area away from telephones and other interruptions, with space for your terminal, and enough desk space to have two or three manuals open at the same time.

Equipment

You will need access to a working RT–11 system. By a *working* system, we mean that:

- The RT–11 monitor program has been transferred from its storage disk to main memory (in other words, the system has been bootstrapped)
- The language processing program for FORTRAN IV or BASIC–11 has been installed and made available for use
- A storage medium—whether it is a disk cartridge, diskette, or magnetic tape—is available for the programs you will create

If you need to bootstrap your system, you may ask a colleague who has experience working with an RT–11 operating system for assistance. Table 1 lists the manuals that provide information on bootstrapping the computer systems on which RT–11 runs.

Table 1.
Manuals with Bootstrapping Procedures

System	Manual
Professional 300	*RT-11 Automatic Installation Booklet: Professional 325/350* *System Release Notes*
MICRO/PDP-11	*RT-11 Automatic Installation Booklet: MICRO/PDP-11*
PDP-11/23-Plus /24 /44	*Introduction to RT-11*, chapter 2 *RT-11 Automatic Installation Booklet: RX02 Diskettes* *RT-11 Automatic Installation Booklet: RL02 Disk* *RT-11 Installation Guide*
PDP-11 (without automatic bootstrapping)	*Introduction to RT-11*, appendix A *RT-11 Installation Guide*

Resources

Although every effort has been made to make *Working with RT–11* a self-contained volume, you may need to refer to the following manuals from the RT–11 documentation set for additional information:

- *Introduction to RT–11*
- *RT–11 System User's Guide*
- *RT–11 System Utilities Manual*
- *RT–11 Software Support Manual*

The documentation to which we refer throughout the text is written for RT–11 version 5.0. We also used a computer system equipped with RT–11 version 5.0 to generate the programs in our examples and practices. If you own a newer version of RT–11, you may also need a copy of the latest *System Release Notes* to determine the difference between your system and the one described here. A list of manuals and books that provide supplementary information is included at the end of this book.

Notations

The following symbols are used in this book to represent specific elements:

⟨KEY⟩	indicates keyboard and keypad keys, their functions, or key combinations
.	indicates the prompt displayed by the system monitor
`COMMANDS`	(uppercase) indicates input
`Prompts`	(upper and lowercase) indicates computer output
[ ]	indicates parts of a command that are optional (the brackets are not part of the command string)

Working with RT–11

1

1

Identifying RT–11 Components

This chapter describes the hardware and software components of a typical RT–11 computer system. The essential hardware components include a processor, a terminal, and a storage medium. The system software consists of the RT–11 operating system, language processors, and application programs. This chapter introduces you to the components of the RT–11 operating system—the monitor, the device handlers, the utility programs, and the support for language processors—as well as the components of the monitor—the Resident Monitor (RMON), the Keyboard Monitor (KMON), and the User Service Routine (USR). It discusses the three types of monitors available on RT–11 systems—the Single Job monitor (SJ), the Foreground/Background monitor (FB), and the Extended Memory monitor (XM). A list of the documentation written for RT–11 is provided near the end of the chapter.

System Hardware

RT–11 is an operating system that will run on a number of computer systems. The minimum hardware configuration for an RT–11 system consists of a PDP–11, LSI–11, or SBC–11 processor, 32 Kbytes of main memory, one terminal, and a disk drive with disks or diskettes for mass storage and for backup. Larger systems may have a clock, more memory, more terminals, and more peripheral devices. Table 2 lists the minimum sets of components required to run RT–11.

The Processor

You can use an RT–11 operating system with a variety of PDP–11 processors, ranging in size from the single board LSI–11 to the PDP–11/44.

You will notice different switches, lights, and buttons on the front panel of each PDP–11 computer. On a computer system running RT–11, these are used only to start the system. All further communication between you and the system is done through the terminal.

The Terminal

You may use either a video or printing terminal as your input and output device. Generally, an RT–11 computer system has only one terminal, called the console terminal, through which all interaction between you and the system takes place. Additional terminals may provide auxiliary message-printing capabilities.

The Storage Device

Your RT–11 system allows you to use both cartridge and floppy disk drives and to have access to more than one disk or diskette on your system at a time.

Table 2.
Minimum Hardware Requirements of an RT-11 System

Processor	*Minimum Memory*	*System Device*	*Storage Device*	*Console Terminal*
Professional 300	256K bytes	RD51	RX50	Professional Integral
MICRO/PDP-11	256K bytes	RD51	RX50	
PDP-11 Unibus 11/04, 11/05, 11/10, 11/20, 11/24, 11/25, 11/35, 11/40, 11/44, 11/45, 11/50, 11/55, 11/60, PDT 11/150	32K bytes	RK05 RK06 RK07 RL01 RL02 RX01 RX02	Magnetic Tape RK05 RL01 RL02 RX01 RX02	VT100 LA12 VT101 LA34 VT102 LA38 VT105 LA100 VT125 LA120 VT131
PDP-11/03 (LSI-11)	32K bytes	RK05 RL01 RL02 RX01 RX02	RK05 RL01 RL02 RX01 RX02	VT100 LA12 VT101 LA34 VT102 LA38 VT105 LA100 VT125 LA120 VT131
PDP-11/23 PDP-11/23- PLUS	64K bytes	RL01 RL02 RX02	RL01 RL02 RX02	

A disk or diskette—also called mass storage medium or volume—provides an area, apart from main memory, to keep information. The information may be the programs that make up the system software, applications programs, programs you create, data needed by a program, the results of a computer operation, or textual information. The RT–11 operating system, for instance, is stored on a storage medium referred to as the system volume. When needed, information from the storage medium is transferred into computer memory.

To access information that is stored, you must insert the storage medium (disk or diskette) into a drive. Each drive is assigned a device name (a mnemonic) and a unit number

(0, 1, 2 etc.). Once the disk is inserted into the drive, the disk drive's symbol identifies the storage volume.

One way to protect information on a disk is to make a copy of it on a second storage volume. The copy, called a backup, insures you against the loss of information. Some storage volumes provide a mechanism that protects information against accidental erasure. This mechanism is generally a switch—on the disk itself or on the drive—that you can set to write-protect or write-enable. You can protect a floppy diskette by covering its write-enable notch with a metallic sticker. (Disks and the files that they may contain are discussed in chapter 3, "Storing Data on Disks.")

Optional Hardware

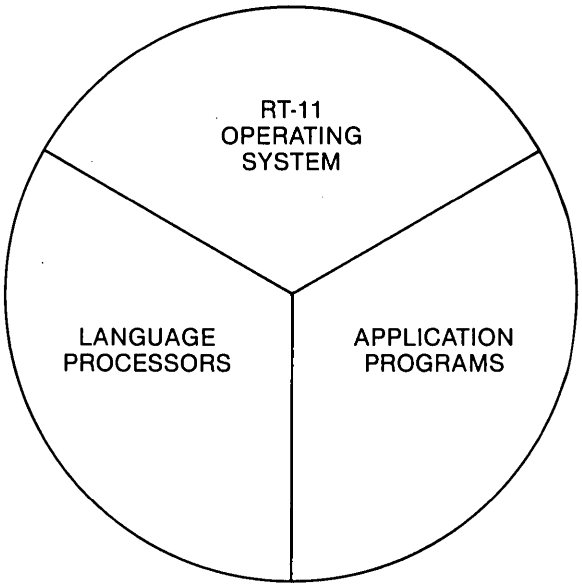
The specific requirements of certain users may dictate the need for additional peripheral devices. For example, computer systems used mainly for program development may need extra storage devices and a high-speed printer. Computer systems used in a laboratory may need graphics display hardware, and computer systems that provide information in conjunction with another kind of computer system will usually require a magnetic tape device, because magnetic tape is a standard storage device across the industry. Smaller PDP–11 systems, such as the MICRO/PDP–11 and the Professional computers, cannot accommodate as much additional hardware as larger systems.

System Software

Your system software is the set of programs that transforms your hardware components into usable tools. Some of these programs store and retrieve data among various peripheral devices. Others perform difficult or lengthy mathematical calculations. Some programs allow you to create, edit, and process application programs of your own, and others handle applications for you.

The system software, as illustrated in figure 1, in-

Figure 1.
System Software



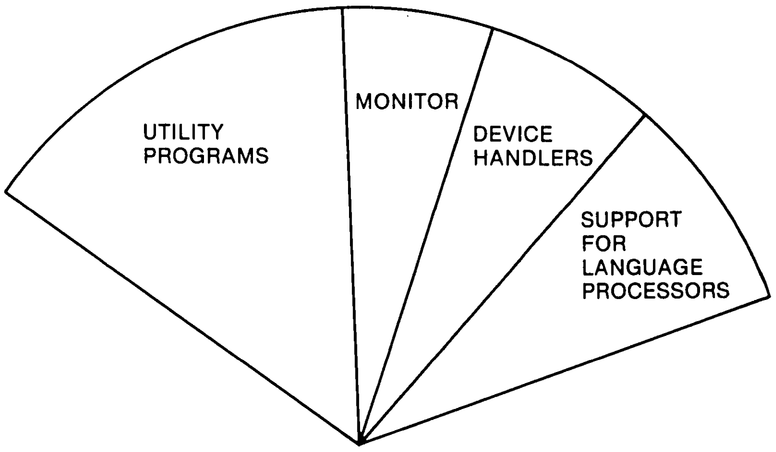
cludes the RT–11 operating system, which is the "intelligence" of the computer system. In addition, your system software probably includes one or more language processors and possibly specific applications as well.

The RT–11 Operating System

The operating system is a collection of programs that organizes all the hardware and software resources of the computer system into a working unit and gives you control. It allows you to create and run programs of your own.

The RT–11 operating system is made up of four types of programs: the monitor program for control of system operation; several device handlers, one program for each of the supported hardware devices; a variety of utility programs for program and data creation and manipulation; and the collections of programs that are necessary to support several programming language processors (see figure 2).

Figure 2.
RT–11 Operating System



The Monitor

Your link with the system hardware and software is the monitor, and the RT–11 monitor provides many different subprograms or routines which work together to perform basic system functions. These functions include the following:

- Acceptance, acknowledgement, and processing of commands that you, or a program, may issue to the system
- Control of all input and output to and from the system
- Timing, or scheduling, of when jobs (programs) should run
- Maintenance of system files
- Checking for abnormal conditions within the system and issuing clear, informative messages
- Control of the way memory is used by the system

The three main parts of the monitor that perform these and other functions are, the Resident Monitor (RMON), the

Keyboard Monitor (KMON), and the User Service Routine (USR).

The Resident Monitor (RMON) provides the console terminal service and central program code necessary for both system and user programs. When RT–11 is started, RMON is automatically loaded from disk storage into processor memory, where it stays until the system is closed down or restarted. The resident monitor is so named because it always remains in computer memory, regardless of system operations. To tell you that it has been loaded and started, RMON issues a message to the terminal.

EXAMPLE

```
RT-11SJ (S) V05.00
```

Note that the message is made up of more than one part.

RT–11	The name of the operating system.
SJ	The abbreviation for Single Job. This describes the specific type of monitor you are using. You may also see FB or XM appear here to indicate that you are using a Foreground/Background or Extended Memory monitor.
(S)	The abbreviation for system generation, the term used to describe the building of an operating system from its component parts. System generation can be performed by those users who wish to create a tailor-made operating system, but RT–11 comes in ready-to-use form.
V05.00	The version number of the monitor currently being used. This number may be different on your system, because as additions or changes are made to the system the number is corrected to show the current revision level.

You will see that the message is followed by a period (.) on the next line. This is called a system prompt or dot prompt and indicates that the monitor is waiting for you to issue a command. The dot prompt is sent to the terminal by the Keyboard Monitor.

The Keyboard Monitor (KMON) controls the interaction between the monitor and keyboard and is the most visible part of the system software from your point of view. Among other services, KMON supplies the monitor command language, a set of command words, that you use to initiate and control system operations. KMON is loaded into memory when needed and processes user commands. Because you will need it only when commands are issued, KMON does not stay in memory but is replaced or "overlaid" by other programs. KMON is swapped in and out of main memory so quickly that you will not notice the activity as you type at the terminal.

The User Service Routine (USR) is used to access information stored on disks and tapes. Once it has finished its work, the USR can be overlaid in the same way as KMON. However, because you will probably handle information kept on a storage medium more often than you will process system commands, you may find it more efficient to keep the USR in main memory at all times. You can instruct the system to keep the USR permanently in memory by using the SET command.

EXAMPLE

```
.SET USR NOSWAP(RETURN)
```

RT–11 provides three different kinds of monitors, each with its own RMON, KMON, and USR. The three monitors are Single Job (SJ), Foreground/Background (FB), and Extended Memory (XM).

The Single Job (SJ) monitor is the simplest and smallest monitor. The resident portion occupies only 4 Kbytes of main memory and therefore leaves most of the memory free for other programs and data. It can support one user program or system utility program and is ideal for smaller systems (sometimes called dedicated systems) which are being used for one specific application, such as scientific data logging or process control. The SJ monitor is not available on the Professional 325 or 350.

A special version of the SJ monitor, the Base Line (BL) monitor, also runs in a minimum configuration of 32 Kbytes of memory, but it does not support optional monitor and device functions. The BL monitor is best suited for very small hardware configurations or for larger configurations in which the application requires minimal executive support.

The Foreground/Background (FB) monitor has all the capabilities of the Single Job monitor as well as its own features. Many applications call for a high priority program, such as data collection, and a lower priority data analysis program. The Foreground/Background monitor allows two programs to run at the same time. The high priority job is known as the foreground job and the low priority job is known as the background job. Since the FB monitor handles multiple jobs, you can work at the keyboard while the system processes another job.

The PDP–11 family contains processors which will support up to 4092 Kbytes of memory. The Extended Memory (XM) monitor allows you to make use of that memory and allows jobs to be 128 Kbytes in size. The facilities of the Foreground/Background monitor and the Single Job monitor are still available with the XM monitor.

Device Handlers

Device handlers are programs that transmit control signals and data to and from your system's peripheral hardware components. These programs check each device for errors, which they then report to the monitor for action. For example, the program which looks after the line printer checks that the printer is on and that it has paper in it. The program also makes sure that any necessary control characters are transmitted and that the characters for printing are valid. You will not come in contact with the device handlers except when the monitor asks you to make a correction, for example, to load paper as a result of the device handler's check.

Utility Programs

RT–11 provides a number of utility programs that allow you to store data on a variety of devices, develop your own programs, and manage the system. Among the utility programs to be discussed are the following:

- An editor, which allows you to create and change memos, programs, and documents
- Debugging programs which help you uncover and correct errors in your programs
- File maintenance programs which allow you to manipulate the information contained in files or on devices
- A librarian, which makes it easy for you to store and retrieve frequently used programming routines
- A linking program, which converts programs into a format suitable for loading and execution
- A source comparison program, which is used to compare text files and to report any differences
- A dump program, which outputs to the terminal or line printer the binary representation of any part of a file

Many of the utility programs may be used by means of simple, easy-to-learn commands.

Support for Language Processors

Two language processor support programs are available as part of the RT–11 operating system. The FORTRAN IV System Subroutine Library (SYSLIB) allows a FORTRAN IV user to write almost all application programs completely in FORTRAN IV with no assembly language coding. The other language processor support program is SYSMAC, the macro library that contains system macros used in assembly language, in this case MACRO–11, programs.

Language Processors

Language processors are translation programs that convert the programs you create at the terminal into a series of binary codes that the computer can understand. They also translate binary codes back into words for output on a video or printing terminal. A language processor exists for every programming language supported by the system, whether it is a high-level language or an assembly language.

Assemblers

Assemblers translate assembly language into object code, a language that the machine can execute. An assembler is a one-for-one translator; that is, each instruction in assembly language becomes an instruction to the computer. The assembly language on PDP–11 computers is called MACRO–11. Though MACRO–11 programs are slower to code and compile than programs written in high-level languages, assembly language gives you more control over factors such as program size and speed of execution.

Compilers

Compilers process high-level languages such as FORTRAN IV and COBOL–11. Whereas assemblers translate one language instruction into one object code, compilers may translate one language instruction into many object codes. For example, a single FORTRAN command may be compiled into twenty or more machine instructions. Though this coding process is faster than that of an assembler, you must relinquish some control over program execution.

Most compilers do not translate the source or language code until it passes through the entire program at least once. Such multipass compilation—called code optimization—allows the compiler to eliminate unnecessary code and to check for errors at many levels.

Interpreters

Interpreters translate instructions written in a high-level language, such as BASIC–11, into a format the computer can interpret. Rather than converting the entire program to object code before running it, an interpreter translates and executes a program on a statement-by-statement basis. Though interpreted programs run slowly, using an interpreter may enhance program development since you can receive an immediate response from the computer when errors in a program are detected.

Applications Programs

The RT–11 operating system supports numerous applications packages. These include an applications package for the standard functions found in most laboratories. A scientific package for FORTRAN IV users provides a large selection of mathematical and statistical routines commonly required in scientific programming. And a graphics support package for BASIC–11 and FORTRAN IV users provides display features such as multiple intensity and blinking vectors (lines), alphanumerics, and points. In addition, during RT–11's ten-year history, scores of real-time and commercial applications have been written by original equipment manufacturers (OEMs) and customers for use with RT–11 systems. Table 3 lists the names of some widely-used applications programs written for RT–11.

Table 3.
Applications Packages for RT-11

	Applications
Word Processing	LEX-11 Glenn A. Barber Associates WPS WP Saturn Word/List Processing MASS-11
Electronic Spreadsheet	Saturn CALC C-Calc CALC-11 Super Comp
Time-sharing	CTS-300 TSX-Plus SHARE-11
Sorting	ZSORT RTSORT
Database Management	RT-DBASE "D" RTFILE SIMILE

System Documentation

Documentation includes manuals that tell you how to use the software and hardware of the computer system. It also includes any source listings of programs that make up the operating system.

Hardware Manuals

Hardware manuals describe the devices in the computer system. RT–11 hardware documentation includes a processor handbook that describes the PDP–11 computer you are using, and a user's guide or maintenance manual for each peripheral device in your computer system. These manuals tell you how to operate the devices and give you special programming information that you may need if you intend to write device drivers or special system software involving the devices.

Software Manuals

Software manuals describe the operating system and the language processors. RT–11 software documentation consists of introductory manuals (intended to be used once and then stored away), console manuals (intended to be used at the computer), and reference manuals (intended to be used at your desk for reference). Table 4 lists the introductory, console, and reference manuals available for RT–11.

Source Listings

Source listings are actual listings of the assembly language codes that make up the RT–11 operating system. These listings are very detailed and generally are needed only if you intend to modify the system software.

Table 4.
Documentation for RT-11

	Manuals
Introductory	*Introduction to RT-11* *System Release Notes*
Console	*System User's Guide* *System Utilities Manual* *System Message Manual*
Reference	*Programmer's Reference Manual* *Software Support Manual* *MACRO-11 Language Reference Manual*

Summary

MINIMUM HARDWARE CONFIGURATION

PDP–11, LSI–11, or SBC–11 Processor
32 KBytes of Main Memory
One Terminal
One Storage Device

COMPONENTS OF SYSTEM SOFTWARE

RT–11 Operating System
Language Processors
Applications Programs

COMPONENTS OF THE RT–11 OPERATING SYSTEM

Monitor
- *Types*
  1. Single Job (SJ)
  2. Foreground/Background (FB)
  3. Extended Memory (XM)
- *Components*
  1. Resident Monitor (RMON)
  2. Keyboard Monitor (KMON)
  3. User Service Routine (USR)

Device Handlers
Utility Programs
Support for Language Processors

2

2 Getting Started

This chapter puts you at the keyboard of your computer terminal to begin using your RT–11 operating system. To become familiar with the keyboard and monitor, you will work with some simple monitor commands. You will learn to issue commands that call up the HELP text and use special characters to stop and start a listing, suppress output, abort a job, and delete text. You will also learn to set the system's clock and date-tracking device.

Working with the Keyboard

All 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. The paper printer and the screen serve the same purpose; they show your input and the system's responses. However, paper output can be saved while screen output is temporary.

You type all commands to the system on the keyboard. Figures 3 and 4 show the layout of the VT100 and the Professional 300 keyboards. The keys for the alphabetic characters, the ⟨SHIFT⟩, ⟨CAPS LOCK⟩, and ⟨TAB⟩ keys work and are arranged in the same way as those on most standard typewriters. Table 5 describes the functions of some keys you will use often.

Table 5.
Keys and Their Functions

Symbol*	Key	Function
⟨RETURN⟩	Carriage Return	Ends a line and moves cursor to the next line Ends certain system commands, transmits them to the processor, and moves the cursor to the beginning of the next line
⟨CTRL⟩	Control	Forms two-key control commands that perform specific functions The system carries out the function as soon as you press ⟨CTRL⟩ and the other key simultaneously
⟨DELETE⟩	Delete	Erases the character to the left of the cursor
⟨LINEFEED⟩	Linefeed	Ends certain system commands, transmits them to the computer, and moves the cursor to the beginning of the next line

*This symbol—the label on or function of a key enclosed in angle brackets—is used throughout the book to represent a key or its function.

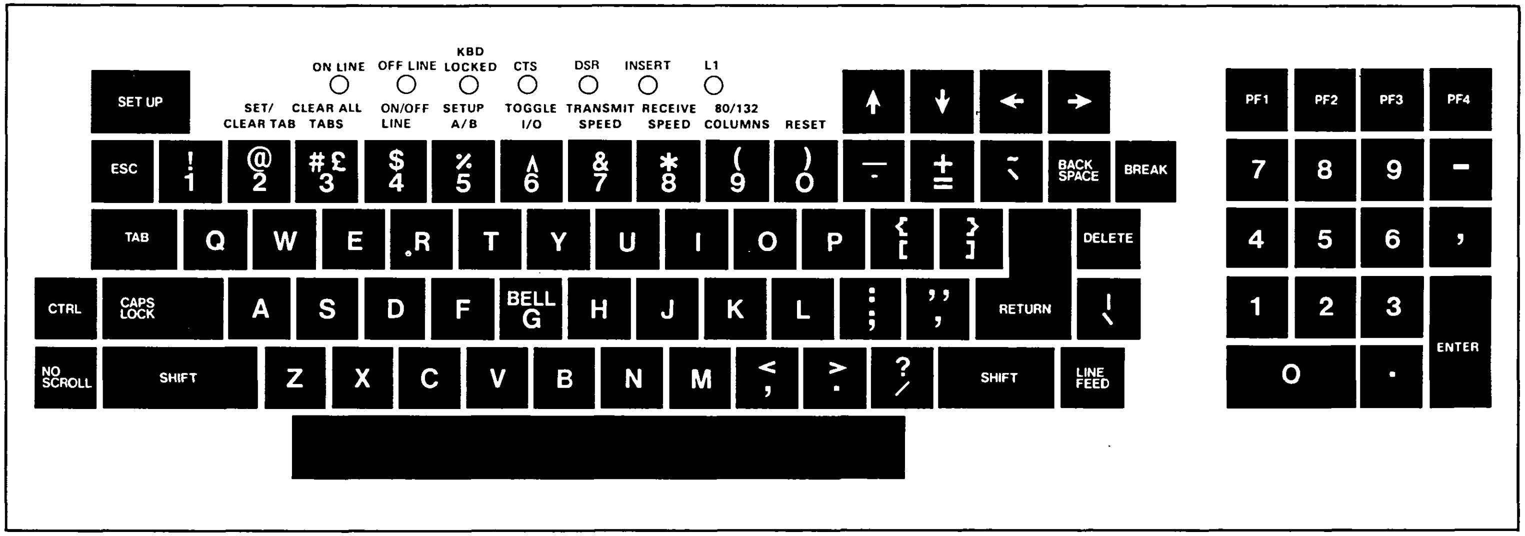


Figure 3. Keyboard Layout of the VT100 Terminal

Monitor Commands

Monitor commands are words or letters you type in at the terminal after the period (.) monitor prompt to start up and control system operations. When you enter a command, you are communicating with the monitor program described in chapter 1.

To issue a monitor command that the computer will accept, you must supply the following information:

- The command for the system operation you want initiated. It may be followed by a slash and a command option which modifies the default operation (the action RT–11 normally takes)
- The input information—the location (indicated by a device name, a file name, and a file type) of the data that is to be processed.
- The output information—the location (indicated by a device name, a file name, and a file type) where the information produced by the operation is to be stored (chapter 3, "Storing Data on Disks," discusses file specifications)

Whenever you type a command, the last key that you press must be the ⟨RETURN⟩ key. ⟨RETURN⟩ will be shown at the end of commands throughout this book to indicate input and to remind you that it is necessary.

Monitor commands may be written in either long format or short format. For long format commands, the system displays prompt messages, which ask for all the information the command needs to be completed. For short format commands, you enter all the information in one command line. The monitor provides prompts only if you leave out essential information. Short format commands are preferred by experienced users, while long format commands are useful for beginners.

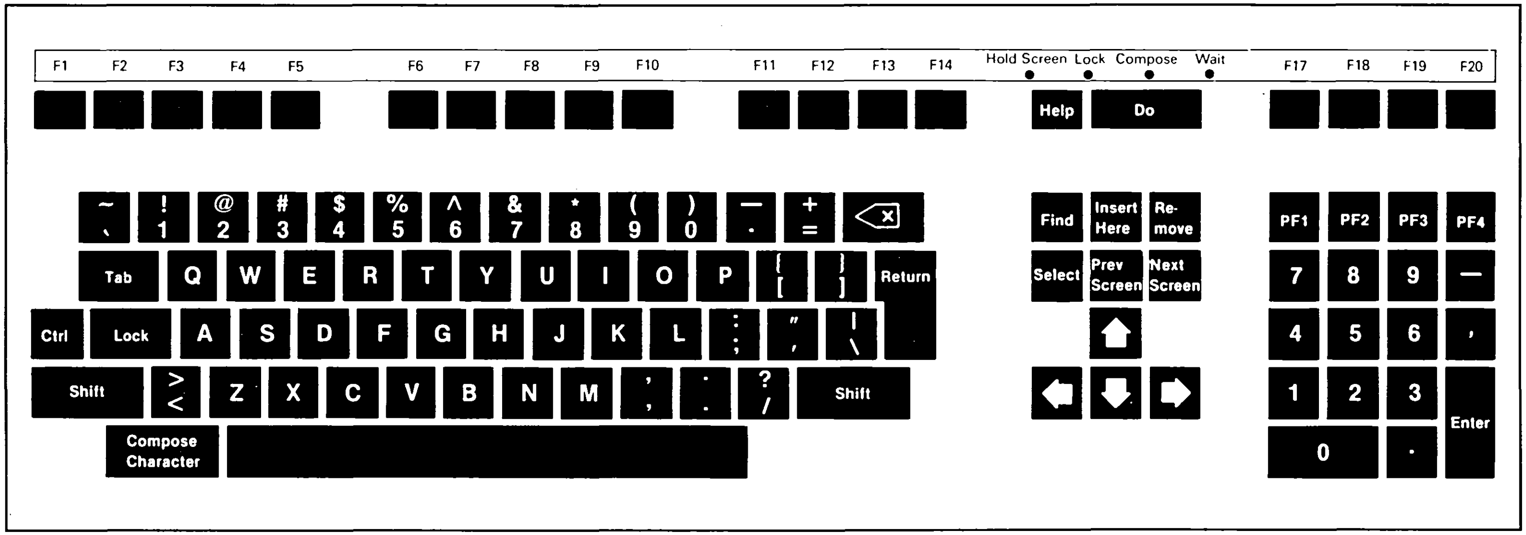


Figure 4. Keyboard Layout of the Professional 300 Terminal

EXAMPLE

The long format of the COPY command is:

```
.COPY⟨RETURN⟩
From? FILE1.TMP⟨RETURN⟩
To   ? FILE2.TMP⟨RETURN⟩
```

The short format of the command is:

```
.COPY FILE1.TMP FILE2.TMP⟨RETURN⟩
```

The meaning of the COPY command will be explained later; for now, just compare the long and short formats of the command. In this chapter we will examine several commands just to see RT–11 in action.

Correcting Typing Mistakes

If you make a mistake while typing a command, you can correct it by using either the ⟨DELETE⟩ key (labeled ⌫ on the Professional keyboard) or the ⟨CTRL/U⟩ key combination.

⟨DELETE⟩ erases characters from right to left, starting with the last character typed. Each time you press ⟨DELETE⟩, it erases one character to the left of the cursor. ⟨DELETE⟩ can only remove characters from the current line of typing, that is, before you press ⟨RETURN⟩.

⟨CTRL/U⟩ deletes all the characters on the current line and moves the cursor to the beginning of a new line so that you can enter the next character or command. ⟨CTRL/U⟩ is displayed on your terminal as ^U.

Practice 2–1

1. If you are using a video display terminal with the Single Job monitor, type:

```
SET TT SCOPE⟨RETURN⟩
```

after the period monitor prompt. SET TT SCOPE tells RT–11 that your terminal is a video rather than a printing terminal.

2. Type the letters:

```
MISTAEK
```

Do not press ⟨RETURN⟩. Once you press the ⟨RETURN⟩ or ⟨LINE-FEED⟩ key, you will not be able to erase the characters.

3. To erase the transposed letters, press:

⟨DELETE⟩⟨DELETE⟩

You may then type in the letters:

```
KE
```

4. Now use ⟨CTRL/U⟩ to erase the word:

```
MISTAKE
```

While you hold down ⟨CTRL⟩, press ⟨U⟩. The two keys should be pressed simultaneously. All the characters will be erased.

Using the HELP Command

RT–11 has a very useful feature called HELP. Whenever you have questions about the system, typing the HELP command may very well provide or direct you to the answer. To see what commands are available, you would type the HELP command followed by an asterisk, press ⟨RETURN⟩, and be provided with a list of commands (shown in figure 5).

You need not memorize the commands you see since the list only shows the kinds of options that are available to you. If you need information about any of the topics listed, you can find it by typing HELP, followed by the name of the topic. For example, typing HELP SHOW will give you more details about the SHOW command (see figure 6).

Figure 5.
HELP Listing

```
ABORT           Terminates, from the system console, a Foreground or
ASSIGN          Associates a logical device name with a physical device
B               Sets a relocation base
BACKUP          Backup/Restore large files or random access devices
BASIC           Invokes the BASIC language interpreter
BOOT            Boots a new system
CLOSE           Makes background output files permanent
COMPILE         Translates source programs
COPY            Copies files
CREATE          Creates a file at a specific block address; extends
D               Deposits values in memory
DATE            Sets or displays the current system date
DEASSIGN        Removes logical device name assignments
DELETE          Removes files from a device or queue
DIBOL           Invokes the DIBOL language compiler
DIFFERENCES     Compares two files and lists the differences
DIRECTORY       Lists device or file directories
DISMOUNT        Disassociates a logical disk assignment from a file
DUMP            Prints formatted data dumps of files or devices
E               Prints the contents of memory on the terminal
EDIT            Invokes the text editor
EXECUTE         Translates, links, and runs a program with one command
FORMAT          Formats and/or verifies a volume
FORTRAN         Invokes the FORTRAN language compiler
FRUN            Loads and starts a foreground program
GET             Loads a memory image file into memory
GT              Enables or disables the VT11 or VS60 display hardware
HELP            Lists helpful information
INITIALIZE      Initializes device directories
INSTALL         Adds a new device handler to the system
LIBRARY         Creates and alters object and macro libraries
LINK            Produces an executable program
LOAD            Makes a device handler permanently resident in memory
MACRO           Invokes the macro assembler
MOUNT           Assigns a logical disk unit to a file
PRINT           Prints files on the line printer
PROTECT         Sets RT-11 file protection status
QUEMAN          Queue control program.
R               Loads and executes a memory image file
REENTER         Starts a program at its reentry address
REMOVE          Removes a device handler from the system
RENAME          Changes the name of a file
RESET           Causes a general system reset
RESUME          Resumes execution of a foreground or system job
RUN             Loads and starts a program
SAVE            Writes memory areas to a file
SET             Controls various system options
SHOW            Displays system hardware and software status
SQUEEZE         Rearranges disk files to collect unused file space
SRUN            Loads and starts a system job
START           Initiates the program in memory
SUSPEND         Stops execution of the foreground or system job
TIME            Sets or displays the system time
TYPE            Outputs files to the terminal
UNLOAD          Removes a resident device handler or FG job from memory
UNPROTECT       Resets RT-11 file protection status
```

Figure 6.
HELP Text for the SHOW Command

```
 HELP SHOW

SHOW            Displays system hardware and software status

  SYNTAX
        SHOW [options]

  SEMANTICS
        SHOW<CR> displays the device assignments; other information
        is displayed by specifying one or more option names.

  OPTIONS
   ALL
        Shows configuration, devices, jobs, and terminals
   CONFIGURATION
        Indicates the monitor version number,SET options,
        hardware configuration,and SYSGEN options
   DEVICES
        Indicates the status and vectors of all device handlers on the
        system
   ERRORS[/options]
        Produces on the terminal a report of all system and device
        errors; valid only if error logging is present.  Options are:
            /ALL         (default)
                Produces the report for all errors
            /FILE[:filnam.typ]
                Specifies the name of the file containing the logged
                errors; defaults to ERRLOG.DAT
            /FROM:[dd:mmm:yy]
                Reports only errors that occurred after the date
                specified
            /OUTPUT:filespec
                Produces the report in the specified file
            /PRINTER
                Produces the report on the line printer
            /SUMMARY
                Produces a summary report
            /TERMINAL    (default)
                Produces the report on the terminal
            /TO:[dd:mmm:yy]
                Reports only errors that occurred before the date
                specified
   JOBS
        Lists the names and status of all loaded jobs
   MEMORY
        Displays the current memory organization in tabular form
   QUEUE[/DEVICE:dev]
        Lists the contents of the line printer (or specified device)
        queue
   SUBSET
        List the current logical disk subsetting assignments in effect
   TERMINALS
        Indicates the status and SET options of all the terminals on the
        system (if a multi-terminal monitor)

  EXAMPLES
        SHOW
        SHOW CONFIGURATION
        SHOW QUEUE
```

Practice 2–2

After the period monitor prompt, enter the command:

```
HELP *(RETURN)
```

and survey the listing to get an overview of the commands used with RT–11.

Using Special Key Combinations

Suppose you wanted to search the HELP * listing for a particular command. You need to be able to stop and start the listing at intervals to do this search. To stop a listing, you can use the ⟨CTRL/S⟩ key combination. To continue the listing, you can use the ⟨CTRL/Q⟩ key combination. (⟨NOSCROLL⟩ on the VT100 terminal or ⟨HOLD SCREEN⟩ on the Professional terminal also allows you to stop and start a listing. You simply press the key once to stop a listing and press it again to continue the listing.)

Another way to stop a listing while it is running is to use the ⟨CTRL/O⟩ key combination, which stops the listing being displayed on your terminal without stopping the program. If you press ⟨CTRL/O⟩ a second time before the listing is completed, the remaining parts of the listing will be displayed. To abort a program while it is running you press ⟨CTRL/C⟩ twice.

You have already used one of the special key combinations, ⟨CTRL/U⟩. If you are using a display terminal, you should enter the command, SET TT NOSCOPE, before you try using other special key combinations, so that listings do not stop when the screen is full. It is not essential to do this to use the special key combinations, but NOSCOPE makes it easier to see the effects of using the key combinations discussed.

Practice 2–3

1. Type:

   ```
   HELP *⟨RETURN⟩
   ```

   after the dot prompt and try to stop and start the listing using the ⟨CTRL/S⟩ and ⟨CTRL/Q⟩ key combination. Remember to hold down ⟨CTRL⟩ while you press ⟨S⟩ or ⟨Q⟩. See if you can stop or start every few lines.

2. Type:

   ```
   HELP *⟨RETURN⟩
   ```

   again following the monitor prompt, and this time stop the listing with ⟨CTRL/O⟩. Compare the effects of using ⟨CTRL/O⟩ to start and stop output with the use of ⟨CTRL/S⟩ and ⟨CTRL/Q⟩, which affect the running of the program and the output.

3. Type:

   ```
   HELP *⟨RETURN⟩
   ```

   Press ⟨CTRL/C⟩⟨CTRL/C⟩ to end the HELP listing.

The effects of all special key combinations discussed in this chapter are shown in table 6.

Table 6.
Control Key Combinations

Key Combination	Function
⟨CTRL/S⟩	Stops the current process
⟨CTRL/Q⟩	Starts or continues the current process
⟨CTRL/O⟩	Suppresses the display of a listing on the terminal
⟨CTRL/C⟩⟨CTRL/C⟩	Aborts a program and returns control to the monitor
⟨CTRL/U⟩	Deletes the line of characters to the left of the cursor

Setting the Date and Time

Setting the date and time allows the system to keep a record of when operations are performed, for instance, when files are created. If you are not sure whether the date has been set, you may type the DATE command followed by a ⟨RETURN⟩. If a date has not been set, an error message will be displayed.

EXAMPLE

```
?KMON-W-No date
```

?KMON stands for a message from the keyboard monitor. W indicates that the message is a warning.

If a date has not yet been set on your machine you may enter today's date in this format:

DATE dd-mmm-yy

EXAMPLE

```
.DATE 28-APR-83⟨RETURN⟩
```

Typing in the current date replaces any previous date.

You can set the time in a similar way. Again, you should test to see if the time has been set by typing the TIME command followed by a ⟨RETURN⟩. If your system does not have a clock, an error message will be produced.

EXAMPLE

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

If your system does have a clock, you may set it to the right time by following the format:

```
TIME hh:mm:ss
```

Typing a new time sets the clock at that time. The system clock keeps track of the current time like any other clock; time, however, is specified in 24-hour notation. So, if it is 4:15 p.m., you would enter 16:15:00.

Practice 2–4

1. Type:

   ```
   DATE<RETURN>
   ```

   If today's date has not been set, type the correct date in the format:

   ```
   DATE dd-mmm-yy
   ```

2. Type:

   ```
   TIME<RETURN>
   ```

   If your system has a clock and the right time has not been set, type it in the format:

   ```
   TIME hh:mm:ss
   ```

Type-Ahead Function

Characters typed at the terminal while another job is running are stored in a place called the input buffer. A buffer is a storage area—often a special register or a designated area of memory—used to hold information being transferred between two devices or between a device and memory. If the job that is running does not expect or require input from the keyboard, the characters remain stored in the buffer for the next job or are interpreted as the next command by KMON when the current job finishes.

Practice 2–5

Type:

```
DIRECTORY⟨RETURN⟩
```

While the directory, listing the files stored on the system volume, is running type:

```
DATE⟨RETURN⟩
```

Notice that the second command is run when the first is completed.

Summary

KEYBOARD MONITOR COMMANDS

DATE	sets the date
HELP	displays information about system features
TIME	sets the time

KEYS AND SPECIAL KEY SEQUENCES

⟨CTRL/C⟩⟨CTRL/C⟩	aborts a program
⟨CTRL/O⟩	suppresses output
⟨CTRL/Q⟩	starts a listing
⟨CTRL/S⟩	stops a listing
⟨CTRL/U⟩	removes the line of characters to the left of the of the cursor
⟨DELETE⟩	removes one character at a time to the left of the cursor

3

3
Storing Data On Disks

When you work with RT–11, you will store the data and the programs you create on disks or magnetic tapes. To use RT–11 efficiently, you need to understand how information is organized on these devices and how you can access and use it.

This chapter discusses naming files by using file specifications and saving time and space by using wildcards. Time is also spent on six simple monitor commands used for file maintenance: COPY, DELETE, RENAME, PROTECT, UNPROTECT, and DIRECTORY.

File Storage Media

Your RT–11 system stores information in a format called a file on disks or magnetic tapes. A file is a collection of codes which represent data or computer instructions. On a disk, files can be accessed in any order, that is, they can be called up from any part of the disk. But on magnetic tape, files must be read in the sequence in which they have been recorded. Disks provide random-access storage whereas tapes provide sequential-access.

Tape Structure

On magnetic tapes, information that identifies a file (file name, type, creation date, and its sequence number) is stored in a file header. The system automatically places a file header at the beginning of each file. When you have finished putting information in a file, the length of the file is recorded automatically at the end of the file. The last file on a tape is followed by an end-of-tape mark. An empty area, where new files may be added, is maintained after the last file on the volume. When a new file is added to the tape, it overwrites the old end-of-tape mark. A new end-of-tape mark is written after the new file. This means that the most recently created file will always be the one before the end-of-tape mark.

Disk Block Structure

The basic unit for storing information on an RT–11 disk is a block. A block is 256 words, roughly 512 alphanumeric characters or one and one-half typewritten pages, in length. On a disk, each file starts at the beginning of a block and the size of a file is always given in numbers of blocks. Each block on an RT–11 disk is numbered starting with block 0. Files are stored in blocks with sequential block numbers, called contiguous blocks. This means that a given RT–11 file occupies one area on the disk instead of being broken up into separate areas.

Block 0 and blocks 2 through 5 (known as boot blocks) contain space reserved for bootstrap programs. Bootstrap programs perform operations such as reading parts of the operating system into main memory. They are found only on the disk that contains the RT–11 system code. All other disks, such as disks that hold your data, have boot blocks but they are empty.

The code in block 0 is read and executed by the hardware boot program each time you turn on your system. The code in block 0, in turn, reads and executes the code in blocks 2 through 5. Block 1, the home block, contains volume identification information—such as the date the volume was created and the name of its owner.

Each disk has a file called the directory. A directory holds information about where files are stored on a disk and where empty areas are so that new files can be created. Each file is listed by its file name and type in the directory. Every file and every empty area is described by its starting block number and by its size in blocks. The directory always starts in block 6. Files are stored in the area immediately following the directory.

File Specifications

Frequently, the first step in processing data is to identify the file to be used. The full address of a file is called the file specification. The full specification for any file has three parts:

1. The name and unit number of the device on which the physical storage medium holding the file is mounted.
2. The file name, one to six alphanumeric characters assigned by the person who created the file.
3. The file type, which tells you about the format or contents of the information held in the file.

For example, if your file specification is DX1:SUMS.MAC, DX1: is the name of the device on which the file is held (in this example an RX01 diskette). SUMS is the program name given by its author, and .MAC is the file type; here it tells you that the program is written in MACRO–11.

Practice 3–1

Assume that you have a file called TEST written in BASIC and stored on the disk held on device DM0. The default file type for files written in BASIC is .BAS. Write the full specification for this file.

Sometimes you may omit the device name or file type from a specification, because certain utility programs assume values for these elements, called default values. Files are assumed to be on the default user disk if you do not give a device name.

Device Names

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.

Physical Device Names

Each hardware device on an RT–11 system is identified by a permanent two-letter mnemonic. The mnemonics are defined in the system software and are recognized and used by the operating system.

For those devices which can have more than one unit or drive, a third character, a number in the range 0 to 7, is used. This number specifies the drive number; if you omit it, the system assumes 0. Thus, the device name for a diskette in RX01 drive unit 0 is DX: or DX0: and for a diskette in RX01 drive unit 1, DX1:.

So that the system can tell the difference between device and file specifications, a colon (:) always follows a device name.

EXAMPLE

```
.COPY DM0:MYFILE.TXT DM1:<RETURN>
.COPY DM0:MYFILE.TXT DM1<RETURN>
```

The first command copies the file MYFILE.TXT, which is on the RK06/07, drive 0, onto the RK06/07, drive 1, and names the file MYFILE.TXT. The second command copies the file MYFILE.TXT to another file called DM1 which is also on the RK06/07, drive 0.

Logical Device Names

Unlike DX:, the name SY: does not represent a specific device. It is, rather, a logical device name that can be assigned to a particular physical device. SY: is used by commands which access the system volume. For example, a command like RUN PROGRAM-NAME (which runs programs from the system volume) assumes that the device to which SY: is assigned contains the system volume. Similarly, DK: is the default logical device name for the system storage volume. You can assign DK: to any kind of storage device. Since DK: represents the default storage volume, you need not specify the device name in commands.

Practice 3–2

To see what physical devices SY: and DK: represent on your system, type the command:

```
SHOW<RETURN>
```

The listing shown will resemble this one:

```
TT (Resident)
DM (Resident)
```

```
  DM0=DK, SY
MQ (Resident)
LD
VM
MM
NL
5 free slots
.
```

You can assign DK: to a specific disk on your system with the following command format:

ASSIGN PHYSICAL-DEVICE LOGICAL-DEVICE

EXAMPLE

```
.ASSIGN DM1: DK:⟨RETURN⟩
```

To verify that the assignment has been made, you type the SHOW command:

```
.SHOW⟨RETURN⟩
```

A listing like the following should appear:

```
TT (Resident)
DM (Resident)
  DM0=SY
  DM1=DK
MQ (Resident)
LD
VM
MM
NL
5 free slots
.
```

In addition to using default logical device names, you may need to assign other logical device names to physical devices for a number of reasons. You use logical device names in programs you write if you cannot predict which physical device will be available for use. When you run the program, you simply assign the logical device names to the physical device names available on your system. A logical device name is made up of any three characters of your choice.

EXAMPLE

```
.ASSIGN(RETURN)
Physical device name? DY1:(RETURN)
Logical device name? VOL:(RETURN)
```

Once the assignment is made, the system recognizes the logical device name VOL: as the device name for your volume.

Logical device assignments are temporary. Thus, you must reassign a logical device each time you start the system. Table 7 lists the physical and default logical device names used with RT–11.

File Types

The file type generally indicates the format or contents of a file. If you do not supply the file type when giving a file specification, the system may assume one of a number of types, depending on the command you have used. This assumed file type is known as the default file type for that command. For example, if you type RUN PROG1, the system assumes you are referring to a file whose specification is PROG1.SAV. .SAV is the default file type for the RUN command. Some commands, like COPY, assume a wildcard (an abbreviated specification) default, so the COPY command will be performed on all files of that name,

Table 7.
Device Names

	Mnemonic	Device
Physical Device Names	DUn:	RD51 Disk, RX50 Diskette on MICRO/PDP-11, RA80 Disk
	DLn:	RL01/02 Disk
	DMn:	RK06/07 Disk
	DWn:	RD50/51 Disk on Professional 300
	DXn:	RX01 Diskette
	DYn:	RX02 Diskette
	DZn:	RX50 Diskette on Professional 300
	RKn:	RK05/RK11 Disk
	MMn:	TJU16 Magtape
	MSn:	TS11 Magtape
	MTn:	TM11 Magtape
	LP:	Line Printer
	LS:	Serial Line
	TT:	Console Terminal
	(n represents the device unit number)	
Default Logical Device Names	SY:	The system volume
	DK:	The default storage volume

whatever their file type. The file types you will use most often are defined in table 8. (A complete list of standard file types is shown in table 3–2 of the *RT–11 System User's Guide*.)

Wildcards

If you are doing the same operation on a number of files with similar file specifications, it may be more efficient to use special symbols called wildcards in the file specification instead of naming each file. When you use a wildcard,

Table 8.
Frequently Used File Types

File Type	File Represented
.BAS	BASIC source file
.FOR	FORTRAN source file
.MAC	MACRO source file
.DBL	DIBOL source file
.BAK	The backup file created by a text editor
.OBJ	The object file—a binary file made up of an assembled or a compiled program
.SAV	The file type given to executable code—the version of a program that is run by the computer
.LST	The listing file produced as output at the line printer
.COM	The command file type
.SYS	The monitor file and handler file type—the file type given to system programs

the operation specified by the command is performed on all files which match the pattern you have provided.

Two symbols are used as wildcards in RT–11. An asterisk (*) can replace any character or a string of characters. A percent sign (%) can replace any single character. If you have a series of files with the same file type suffix, like SUMS.BAS, WORK.BAS, and GAMES.BAS, for instance, you could use the notation *.BAS to mean all of these files.

The percent sign (%) is used when file specifications differ in one character only. For example, if you want to perform an operation on files called TEST1.BAS and TEST2.BAS, you can use the wildcard notation TEST%.BAS to represent these files. If there is also a file called TEST3.BAS, the wildcard will access it, too.

You can use wildcards in place of file names, file types, or characters in file names or file types. You can also use wildcards to represent part of a file name. For example, US*.* specifies all files whose names begin with "US" on the current volume. The wildcard *.*, on the other hand, will access all files on the current storage volume. You cannot, however, use wildcards to specify devices.

Factoring

Factoring is another method of specifying a number of files without typing in the name of each file individually. For example, TEST(A,B,C).MAC is equivalent to TESTA.MAC, TESTB.MAC, TESTC.MAC.

Practice 3–3

You have the following files:

1. TESTM.MAC
2. TESTF.FOR
3. TESTB.BAS
4. TESTD.DBL
5. TENTM.MAC
6. TENTF.FOR
7. TENTB.DBL
8. TINTM.MAC
9. TEAKM.MAC

Which of the following file specifications selects 1, 2, 3, 5, 6, 7, but not 4, 8, or 9?

a. TE%T (M,F,B).*

b. T*.(MAC,FOR,DBL)

c. *(M,F,B).*

d. TE%T*.*

e. TE*.(MAC,FOR,DBL)

File Maintenance Commands

When you receive your RT–11 system, the system volume 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. However, since the system volume serves as the default storage volume for all system operations (unless the name DK: was assigned to another volume), you will discover that it acquires many additional files during normal use. For example, when you create files with the keypad editor, they are stored on the system volume; when you edit files, the keypad editor automatically saves the original on the system volume; and many utility programs create output and listing files as part of their normal operation. By the time you finish an average session of computer operations, several new file names may have been added to the directory of your system volume.

To avoid having your system volume become full and its directory cluttered with the names of files for which you have no use, you should perform regular file maintenance operations as you use the system. That is, you should update and transfer copies of your important files to other storage volumes for safekeeping and you should delete from your system and storage volume directories the names of files you no longer need.

The RT-11 operating system provides a number of monitor commands for this purpose. These commands activate the RT–11 utility programs called PIP.SAV, DUP.SAV, and DIR.SAV, which allow you to transfer and erase files. Using the monitor commands introduced in this chapter is one way to maintain your system and storage volume.

DIRECTORY: Listing the Files on a Storage Volume

Both your system volume and your storage volume have directories, which are compiled lists of all the files stored on the volume. The DIRECTORY command elicits lists of information about a device, a file, or a group of files. The DIRECTORY command does not prompt you for information. If you do not specify a device name or a file name and a file type, the system lists information about all the files on the default storage device.

Monitor commands may be abbreviated to the minimum number of characters necessary to define a command uniquely. For instance, instead of typing the command DIRECTORY—as you did in practice 2–5—you may truncate the command to DIR. Table 9 lists the minimum abbreviation for each monitor command. In general, abbreviating to three letters gives a unique command.

Table 9.
Monitor Command Abbreviations

Monitor Command	Abbreviation	Monitor Command	Abbreviation
ABORT	AB	INSTALL	INS
ASSIGN	AS	LIBRARY	LIB
BACKUP	BAC	LINK	LIN
BASIC	BAS	LOAD	LO
BOOT	BO	MACRO	MAC
CLOSE	CL	MOUNT	MO
COMPILE	COM	PRINT	PRI
COPY	COP	PROTECT	PRO
CREATE	CR	REENTER	REE
DATA	DA	REMOVE	REM
DEASSIGN	DEA	RENAME	REN
DELETE	DEL	RESET	RESE
DIBOL	DIB	RESUME	RESU
DIFFERENCES	DIF	RUN	RU
DIRECTORY	DIR	SAVE	SA
DISMOUNT	DIS	SET	SE
DUMP	DU	SHOW	SH
EDIT	ED	SQUEEZE	SQ
EXECUTE	EX	SRUN	SR
FORMAT	FORM	START	ST
FORTRAN	FORT	SUSPEND	SUS
FRUN	FR	TIME	TI
GET	GE	TYPE	TY
HELP	H	UNLOAD	UNL
INITIALIZE	INI	UNPROTECT	UNP

Practice 3–4

Get a directory listing of all the files on your storage device with file names ending in the letters ST or TS by typing the command:

```
DIR *(ST,TS).*<RETURN>
```

You can use the DIRECTORY command to obtain a list of files on magnetic tape too, even though tapes have no directory as such. When producing a directory listing from magnetic tape, the utility program reads through the whole tape displaying each file header it encounters.

COPY: Making Copies of Files

The COPY command transfers the contents of one file to another file, the contents of a number of files to a single file, the contents of files from a large volume to several smaller volumes, the bootstrap code to a volume, or the contents of one device to another device. It instructs the system to duplicate the file that you indicate as input, then gives the new file the name and type that you specify as output. The original version of the file is unaffected.

You saw the long and short formats of this command in chapter 2, but they are repeated here to refresh your memory.

EXAMPLE

Long format:

```
.COPY<RETURN>
From? INPUT.TXT<RETURN>
To  ? OUTPUT.TXT<RETURN>
```

Short format:

```
.COPY INPUT.TXT OUTPUT.TXT<RETURN>
```

Practice 3–5

One of the files provided with your RT–11 operating system is called DEMOED.TXT. Use the long format of the COPY command to copy the contents of the DEMOED.TXT file into a file called PRACT.TXT. The following exchange between you and the system should take place:

```
.COPY<RETURN>
From? DEMOED.TXT<RETURN>
To   ? PRACT.TXT<RETURN>
```

RENAME: Changing the Name of a File

You can use the RENAME command if you want to change the file name or the file type. It changes the name or type of a file in the directory without changing or moving the contents of the file. Thus, when you give a RENAME command the storage device indicated in the input and output portion of the command should be the same.

EXAMPLE

Long format:

```
.RENAME<RETURN>
From? INPUT.TXT<RETURN>
To   ? OUTPUT.TXT<RETURN>
```

Short format:

```
.RENAME INPUT.TXT OUTPUT.TXT<RETURN>
```

Practice 3–6

1. Use the short format of the RENAME command to rename the file PRACT.TXT to BILL.TXT. Type:

```
RENAME PRACT.TXT BILL.TXT<RETURN>
```

2. Call up the directory to see that PRACT.TXT is no longer in the directory by typing the command:

```
DIR PRACT.TXT<RETURN>
```

DELETE: Erasing a File Name from the Directory

Once copies of your important files are stored on a storage volume, you may want to delete from the system volume—or any storage volume—the files you no longer need. When you issue the DELETE command, you tell the system to remove information about the file you specify from the volume's directory. In doing so, the space that the file occupies on the volume becomes available for reuse.

You can use wildcards when issuing the DELETE command, but you must do this with caution as file specifications containing wildcards may match a large number of files. The system will ask for confirmation before it deletes files when there are wildcards in the file specification.

Practice 3–7

Delete BILL.TXT using the long format of the command. Type:

```
DELETE<RETURN>
```

Wait for the monitor prompt:

```
Files?
```

Type:

```
BILL.TXT<RETURN>
```

PROTECT and UNPROTECT: Preventing Accidental Deletion of a File

If you have files that are valuable or important, you may want to use the PROTECT command to prevent accidental deletion. The short command format is PROTECT FILE.TXT. When you protect a file, you will not be able to delete it without giving the UNPROTECT command.

You can specify as many as six input files (separated with commas) in the PROTECT or UNPROTECT command line. As with the DELETE command, you can also use the wildcard construction. Once you have protected a file, its protected status will show up in the directory listing with a "P" next to the block size number of the file's directory entry.

EXAMPLE

```
.PROTECT<RETURN>
Files? DEMOF1.FOR<RETURN>
.DIR DEMOF1.FOR<RETURN>
19-Apr-83
DEMOF1.FOR 2P 12-Dec-82
1 File, 2 Blocks
823 Free blocks
```

If you try to delete a protected file you will see an error message:

```
?PIP-W-Protected File FILE.TXT
```

Practice 3–8

1. Unprotect the file SY:DEMOED.TXT.
2. Use the PROTECT command to return the file to its original status.
3. Use the DIR command to see how its protected status is shown.

Summary

A FILE SPECIFICATION INCLUDES

Device name
File name
File type

FILE MAINTENANCE COMMANDS

COPY	transfers the contents of one file to another file
DELETE	erases a file name from the directory
DIRECTORY	lists the files on a storage volume
PROTECT	prevents accidental deletion of a file
RENAME	changes the name of a file
UNPROTECT	removes the protection status of a file

References

Introduction to RT–11. Chapters 4 and 7 discuss entering and using file maintenance commands.

RT–11 System User's Guide. Chapter 4 discusses and lists keyboard commands alphabetically. Appendix A gives command abbreviations.

RT–11 Software Support Manual. Chapters 8 and 9 discuss file format and storage.

Solutions to Practices

3–1. DM0:TEST.BAS
3–3. (a) TE%T (M,F,B).*
3–8. (1) UNPROTECT SY:DEMOED.TXT
(2) PROTECT SY:DEMOED.TXT
(3) DIR SY:DEMOED.TXT

4

4 Using KED to Edit Text Files

A text editor is a utility program used to create and modify files of printable characters. Text files contain the letters, numbers, and symbols you type in at your keyboard; they may be programs, data to be used in programs, reports, memos, or sequences of monitor commands.

The text editors available on RT–11 include a keypad editor KED, and a line editor, EDIT. If you have a video display terminal, you will use a keypad editor instead of the line editor to create and edit text files.

In this chapter you will learn to use KED to perform the following operations: create, edit, and inspect files; move the cursor to any position within your file; search for a word or character; insert, delete, and restore text in files. You will also learn to set a select range and use the CUT and PASTE commands to change the position of text within your file and leave the keypad editor with or without saving the changes you made when editing.

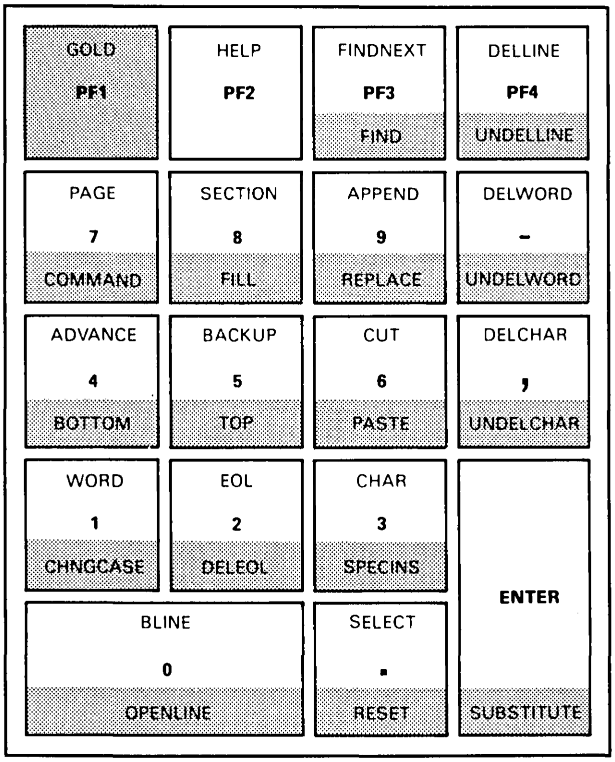
(If you do not have a video display terminal, refer to chapter 6 in the RT–11 System User's Guide *for a discussion of EDIT, the text editor for printing terminals.)*

Editing on a Display Terminal

The keypad editor, also known as a screen editor, allows you to move freely within a text file and see immediately the corrections you are making.

The keypad editor allows you to edit using keys with special functions. On a VT100 terminal, these keys are found on or near the numeric keypad. If you are using a Professional 300 computer, you will find the function keys on the numeric and editor keypads. Figure 7 shows the functions of the keys on the numeric keypad; the functions are the same on the VT100 and Professional 300 terminals. The alphanumeric characters or symbol at the center of a key is the label that actually appears on each key. The term above

Figure 7.
Keypad Functions on the VT100 and Professional 300 Terminal



the label or the label itself describes the function the system performs when you press the key. The term below the label represents the function the system performs when you press ⟨GOLD⟩ (labeled PF1) first. You can use the main keyboard to type in characters as usual, but in this chapter you will concentrate on the function keys and some control characters that have special functions within the keypad editor.

You create and edit text files more often than you perform any other system operation. To facilitate these processes, two RT–11 editor system utility programs, EDIT.SAV and KED.SAV, are stored as part of the RT–11 operating system on your system volume. On all RT–11 systems, EDIT is the default text editor when you start the system. This means that when you issue the monitor command EDIT, the EDIT.SAV text editor program is run.

Practice 4–1

If you have a video terminal, set KED as your default editor by typing the command:

```
SET EDIT KED⟨RETURN⟩
```

Starting the Keypad Editor

The monitor command that starts the keypad editor is EDIT/KED. Since KED is already your default editor, you can simply enter the command, EDIT. The system prompts you for a file name after you issue the command.

Creating Files

The RT–11 editor uses an area in main memory reserved for text that you are typing in for the first time or editing. This area is called the text buffer. When you create a file, the characters you type in at the keyboard are transmitted to the text buffer. When you have finished typing text into the file, the file is transferred from the text buffer to an output file for storage. The monitor command to use the keypad editor to create a file is:

EDIT/CREATE FILENAME

EXAMPLE

```
.EDIT/CREATE TEST.TMP⟨RETURN⟩
```

The name you select for your file should not be the same as a file which you have created previously in your directory. If you try to repeat a name, an error message will be produced.

EXAMPLE

```
KED-W-Output file exists -Continue (Y,N)?
```

The warning message asks you for a yes or no response. If you type in Y, meaning to continue, the original file will be erased.

Saving Files and Leaving the Keypad Editor

When you have created or edited a file and want to save it as you leave the keypad editor, press the ⟨GOLD⟩ key followed by ⟨COMMAND⟩. Pressing these function keys tells the keypad editor that you want to enter a command. The editor, in turn will display the prompt:

```
Command:
```

You should then type the word EXIT and press ⟨ENTER⟩ on your keypad. The system will transfer the information in the text buffer into an output file and bring you out of the keypad editor mode and into the RT–11 monitor command mode.

EXAMPLE

```
<GOLD><COMMAND>
Command: EXIT<ENTER>
.
```

To leave the keypad editor and save the file but not the edits made to it, you type QUIT instead of typing EXIT when the editor prompts you for a command.

Practice 4–2

1. Select a file name, for example TEXT.TXT, and create a file using the keypad editor. Type:

   ```
   EDIT/CREATE TEXT.TXT<RETURN>
   ```

   If you have typed in the command correctly, you will see the cursor positioned on a special symbol, a filled block, known as the end-of-file marker.

2. Type in approximately five lines of text (input a short passage from this book for example). Do not worry about mistakes at this time.

3. When you have finished typing in the text, press <GOLD> and then <COMMAND>. After the system prompt, type:

   ```
   EXIT
   ```

   and press <ENTER>. Your file TEXT.TXT is now stored and you should see the dot monitor prompt.

Editing Files

To edit a file that you created previously, use the monitor command:

```
EDIT FILENAME
```

When you issue the EDIT command, the system copies the file you specify (known as the input file) and transfers the copy to the text buffer where you can modify the material. When you are satisfied with the edits made to the text in the buffer, you may issue either the EXIT or QUIT command.

If you use the EXIT command, the edited text in the buffer is copied into the file you specified (now known as the output file); the edited text replaces the original text in the file.

If you typed in QUIT rather than EXIT when the system prompts for a command, the system returns to the monitor command mode without transferring the text in the buffer to the output file on a storage medium. The edited text in the buffer is lost and the original text remains stored in the file.

If you want to store the edited text in a file with a different name, you can use the /OUTPUT option to the EDIT command. By specifying a different name for the output file, KED will store the edited text in a new file and leave the original text in the input file unchanged. The format for this command is:

```
EDIT/OUTPUT:OUT-FILENAME IN-FILENAME
```

When you edit a file, RT–11 automatically saves the original version as a backup file. It copies the original text into a file using the same name and assigns the .BAK file type to the backup file. For example, if you edited a file called TEST1.MAC the backup copy of the file would be called TEST1.BAK.

Inspecting Files

The keypad editor also allows you to examine files without making any changes in them. To do this you use the EDIT command with the /INSPECT option. The format is:

```
EDIT/INSPECT FILENAME
```

Using Function Keys

Getting Help in the Keypad Editor Mode

When you are using the keypad editor, you can get help by pressing ⟨HELP⟩. This key is labeled PF2 on the VT100 and Professional keyboards (see figure 7). The key labeled HELP on the top row of the Professional keyboard does not work. When we refer to ⟨HELP⟩ in the remainder of this book, we are referring to the PF2 key.

When you press ⟨HELP⟩ you will see a diagram of the keypad, which shows you the position of the functions available. You can also get messages explaining errors by pressing ⟨HELP⟩. When you use a keypad function incorrectly, a bell rings. If you press ⟨HELP⟩ after the bell has sounded, instead of the keypad display, you will be given an explanation of your error.

By pressing ⟨HELP⟩ twice, you can get a listing and description of all the commands and functions available to the keypad editor (see table 10). To enter one of these commands, you must press ⟨GOLD⟩ ⟨COMMAND⟩ and type in the command after the prompt. When you have finished entering the command, press ⟨ENTER⟩.

Using the GOLD Key

When used by itself, ⟨GOLD⟩ (labeled PF1) has no effect. Instead, it enables the alternate function of other keypad keys to work. Except for ⟨HELP⟩ and ⟨GOLD⟩, you will see that all the other keys on the keypad have two functions. The two functions available as well as the label on each key of the VT100 and Professional 300 numeric keypad are shown in figure 7. To use the lower function on a key you must press ⟨GOLD⟩ followed by the function key.

By using the number keys on the keyboard (not those on the keypad) and ⟨GOLD⟩, you can repeat a function any number of times. Press ⟨GOLD⟩ and the number of times you want to repeat a function before you specify the function. For example, ⟨GOLD⟩7⟨DELCHAR⟩ deletes seven characters to the right of the cursor.

Table 10.
Summary of Commands and Keyboard Functions

CLEAR PASTE	Clears the paste buffer
CLOSE	Closes the auxiliary output file
EXIT	Closes all open files
FILL	Reformats the select range to fit within the current right margin
INCLUDE nnn PAGES	Copies pages from the auxiliary input file
INCLUDE nnn LINES	Copies text lines
INCLUDE REST	Copies the remainder of the file
LEARN	Begins recording a sequence of commands and functions as an editor macro
LOCAL [starting-value [increment]]	Renumbers MACRO-11 local symbols
[OPEN] INPUT FILESPEC	Opens an auxiliary input file
[OPEN] OUTPUT FILESPEC	Opens an auxiliary output file
PURGE	Purges the auxiliary output file
QUIT	Purges all open files
SET [ENTITY] PAGE nnn [LINES]	Defines a page by line count
SET [ENTITY] PAGE "string" or 'string'	Defines a page by marker string of one or more characters
SET [ENTITY] SECTION nnn [LINES]	Defines a section by line count
SET [ENTITY] SECTION "string" or 'string'	Defines a section by a marker string of one or more characters
SET QUIET	Sets the video terminal to signal with the reversed background
SET NOQUIET	Sets the video terminal to signal with the terminal bell
SET [SCREEN] 80	Sets the video terminal to 80-column width
SET [SCREEN] 132	Sets the video terminal to 132-column width
SET [SCREEN] LIGHT	Sets the video terminal to light background
SET [SCREEN] DARK	Sets the video terminal to dark background
SET [SEARCH] GENERAL	Matches alphabetically regardless of case

Table 10. Continued

SET [SEARCH] EXACT	Matches alphabetically and requires the same case
SET [SEARCH] BEGIN	Leaves the cursor at the beginning of the target
SET [SEARCH] END	Leaves the cursor at the end of the target
SET [SEARCH] BOUNDED	Limits searching to a page
SET [SEARCH] UNBOUNDED	Allows searching beyond the current page
SET TABS [indent]	Enables structured tabs
SET NOTABS	Enables structured tabs
SET WRAP [nn]	Sets the right margin and enables word wrap
SET NOWRAP	Disables word wrap
SKIP nnn PAGES	Skips pages within the auxiliary input file
SKIP nnn [LINES]	Skips text lines
SKIP REST	Skips the remainder of the file
TABS ADJUST ({±})nnn	Changes indentation
WRITE nnn PAGES	Writes pages (as currently defined) to the auxiliary output file
WRITE nnn [LINES]	Writes text lines
WRITE REST	Writes the remainder of the file
WRITE SELECT	Writes the select range
⟨CTRL/C⟩	Cancels the command or function that is being processed
⟨CTRL/U⟩	Erases one line to the left
⟨CTRL/W⟩	Repaints the screen
⟨CTRL/Z⟩	Cancels editor prompts
⟨DELETE⟩	Erases one character to the left
⟨LINEFEED⟩	Erases one word to the left
⟨GOLD⟩nnn	Repeats the next function nnn times
⟨GOLD⟩⟨S⟩	Stops recording the macro
⟨GOLD⟩⟨X⟩	Executes the macro, as currently recorded
⟨GOLD⟩⟨E⟩	Increases the tab level-counter
⟨GOLD⟩⟨D⟩	Decreases the tab level-counter
⟨GOLD⟩⟨A⟩	Aligns the tab level-counter to the cursor

Using the Cursor Keys

The cursor keys on all VT100 and Professional 300 keyboards are clearly marked with arrow symbols. The right ⟨→⟩ and left ⟨←⟩ arrow keys move the cursor, character by character, forward and backward through the file. The right arrow key ⟨→⟩ moves the cursor to the right and from the end of one line to the beginning of the next line. The left arrow key ⟨←⟩ moves the cursor to the left and from the beginning of a line to the end of the preceding line. Instead of moving the cursor character by character, you can use ⟨GOLD⟩ along with either the left or right arrow key to move the cursor directly to the beginning or to the end of a line.

The up ⟨↑⟩ and down ⟨↓⟩ arrows move the cursor, line by line, vertically through a file. The up and down arrows move the cursor directly up and down, keeping it in the same vertical column of text. When the text line does not reach the column the cursor happens to be in (at the end of a paragraph, for instance) the cursor moves back to the column where the text line ended.

Practice 4–3

Find the arrow functions on your keyboard. Use the EDIT/INSPECT command to gain access to the file you just created and use the arrow functions to move the cursor up and down and then left and right within your file. When you are familiar with how the functions work, use the EXIT command to leave the file.

Moving the Cursor by Units of Text

The cursor can also be moved by using other keypad functions. You set the direction in which you want the cursor to move by pressing ⟨ADVANCE⟩ or ⟨BACKUP⟩. ⟨ADVANCE⟩ moves the cursor forward through the file. ⟨BACKUP⟩ moves the cursor backward through the file.

After you have set the directional mode by pressing ⟨ADVANCE⟩ or ⟨BACKUP⟩, you can instruct the editor to perform a number of functions in the direction indicated. (The arrow functions are not affected by the directional mode. They always move the cursor in the directions indicated on the keys.)

The size of the unit the cursor moves is set with the following functions: ⟨CHAR⟩ moves the cursor, character by character; ⟨WORD⟩ moves the cursor, word by word; ⟨EOL⟩ moves the cursor to the end of a line; and ⟨BLINE⟩ moves the cursor to the beginning of a line.

Depending on the directional mode, ⟨CHAR⟩ advances or backs up the cursor one character, and ⟨WORD⟩ advances or backs up the cursor to the first character of a word. Some keyboards may not have the CHAR function.

Both ⟨EOL⟩ and ⟨BLINE⟩ move the cursor in units of a line. ⟨EOL⟩ moves it to the end of the next line, in the direction set by ⟨ADVANCE⟩ or ⟨BACKUP⟩. ⟨BLINE⟩ moves the cursor to the beginning of the next line, again in the direction you have set.

To move the cursor directly to the beginning or end of a file, you can use ⟨GOLD⟩ with ⟨TOP⟩ or ⟨BOTTOM⟩.

Practice 4–4

1. Use the EDIT command to gain access to one of your files. Press ⟨ADVANCE⟩ to set the direction of the cursor. Try moving the cursor through your text wiih ⟨CHAR⟩ and with ⟨WORD⟩. Then try positioning the cursor at the beginning and end of lines using ⟨EOL⟩ and ⟨BLINE⟩.
2. Change the direction of movement using ⟨BACKUP⟩. Try moving the cursor by character, word, and line.
3. Press ⟨GOLD⟩⟨TOP⟩. When the cursor has reached the beginning of the file and stopped, press ⟨GOLD⟩⟨BOTTOM⟩. You may want to practice using the keypad functions mentioned until you become familiar with them.

Searching for Characters and Words

The FIND function allows you to search your file for a specific character or word. The search will start at the point where the cursor is positioned. If you want to search the complete file, then position the cursor at the beginning of the file by pressing ⟨GOLD⟩⟨TOP⟩.

When you press ⟨GOLD⟩⟨FIND⟩, you will receive the following prompt at the top of the screen:

```
Model:
```

You then type in the character or word you want to find and complete the command with ⟨ADVANCE⟩ or ⟨BACKUP⟩ to specify the direction of the search. The character or word you supply in response to this prompt is known as the search target.

If you want to search for the same word again, press ⟨FINDNEXT⟩. The search will take place in the direction set by ⟨ADVANCE⟩ or ⟨BACKUP⟩ when you used ⟨FIND⟩. If you try to continue the search beyond the beginning or end of the file, you will hear a bell. Pressing ⟨HELP⟩ will give you the error message:

```
Search finds end of file
```

Practice 4–5

Use the EDIT command to get access to a file and press ⟨GOLD⟩ ⟨FIND⟩. Search your file for a word, such as *the*, or a character that appears frequently. Use ⟨FINDNEXT⟩ to continue the search. Try searching both backward and forward through the file. When you are familiar with the FIND function, exit from the file.

Inserting, Deleting, and Restoring Text

Inserting Special Characters

When you insert material into a file that you are creating or editing, you simply type the text on the keyboard just as you would on a typewriter. You cannot however, insert some characters by simply typing them on the keyboard. To insert special characters, such as ESCAPE or LINEFEED, you can use the SPECINS function key. The SPECINS function inserts the character when you specify the character's ASCII code in decimal. When you use the sequence, ⟨GOLD⟩integer⟨GOLD⟩⟨SPECINS⟩, the keypad editor interprets the integer you type as a decimal value, evaluates the seven low order bits, and inserts the corresponding ASCII character. Except for the NULL character, you can use the SPECINS function to insert any character in ASCII.

EXAMPLE

⟨GOLD⟩27⟨GOLD⟩⟨SPECINS⟩

inserts the ESCAPE character (ASCII octal 033, decimal 27).

Inserting Blank Lines

If you want to insert a blank line in your file, you can either press ⟨RETURN⟩ or press ⟨GOLD⟩ with ⟨OPENLINE⟩ on the keypad. ⟨GOLD⟩⟨OPENLINE⟩ inserts a blank line to the right of the cursor. Characters that were originally to the right of the cursor move down one screen line, and the keypad editor scrolls all lines below the cursor downward.

Practice 4–6

1. Use the EDIT command to gain access to a file. Insert blank lines in the text first by pressing ⟨RETURN⟩ and then by pressing ⟨GOLD⟩⟨OPENLINE⟩.
2. Now use the OPENLINE function to insert more than one line. Press ⟨GOLD⟩, type the number of lines you want inserted (2 for instance), and then press ⟨GOLD⟩⟨OPENLINE⟩. When you are familiar with the function use the EXIT command to store your file.

Deletions in your file can be done in three ways, by one character at a time, by one word at a time, or by an entire line at a time.

Deleting Characters

⟨DELCHAR⟩ on the keypad erases the character at the current cursor position. ⟨DELETE⟩ on the keyboard erases the character preceding the cursor.

Deleting Words

The keypad editor considers any text enclosed in spaces, tabs, or new lines as a word. There are two methods of deleting a word: ⟨DELWORD⟩ on the keypad erases forward from the cursor to the first character of the next word. When you use ⟨DELWORD⟩ to erase a word you also erase any spaces before the next word. ⟨LINEFEED⟩ on the keyboard erases backward from the cursor through the first character of the current or preceding word.

Deleting Lines

⟨DELLINE⟩ on the keypad erases forward from the cursor to the beginning of the next line. When the operation is complete, the cursor is at the beginning of the next line.

⟨DELEOL⟩ erases forward from the cursor to the end of the current line; the cursor remains at the end of the current line.

⟨CTRL/U⟩ (holding down ⟨CTRL⟩ on the keyboard while pressing ⟨U⟩) erases back to the beginning of the previous line.

Practice 4–7

1. Use EDIT to get access to a text file. Type in a few sentences and erase characters from them using ⟨DELCHAR⟩ and then ⟨DELETE⟩.

2. Practice using ⟨DELWORD⟩ and ⟨LINEFEED⟩ to erase words. Look at the position of the cursor after each deletion.
3. Try the three ways of deleting a line of text. Look at the cursor position on completion of each operation. When you have finished using the delete functions, use EXIT to return to the monitor command mode.

Restoring Text

The keypad editor uses a character buffer, a word buffer, and a text line buffer to store the last character, word, or line that you erase. If you find you have erased a character, word, or line by mistake you can use either the UNDELCHAR, UNDELWORD, or UNDELLINE functions to copy the contents of the buffer back into your file. ⟨GOLD⟩⟨UNDELCHAR⟩ on the keypad restores a character to the file. ⟨GOLD⟩⟨UNDELWORD⟩ on the keypad restores a word to the file. And ⟨GOLD⟩⟨UNDELLINE⟩ on the keypad restores a line of text to the file.

Practice 4–8

Use EDIT to get access into a file. Try deleting and restoring text in your file in units of characters, words, and lines. (Remember that you can only restore text that you just deleted; the latest deletions replace previous deletions in the buffer.)

Editing Sections of Text

When you are editing a file, you may find that you want to move or modify sections of the text. For example, you have the sentence, "That is the question To be or not to be," and want to move the whole phrase "To be or not to be" and place it before the phrase "That is the question."

The keypad editor provides you with functions to do this, as well as functions to copy sections of text or remove them. You need to indicate to the editor the section of text you want to copy, move, or replace. To do this you must establish a select range. To set a select range, you press ⟨SELECT⟩. The range begins at the position of the cursor. Moving the cursor to the right or left causes text to be included in the select range. A left select range includes the characters to the right of the cursor. A right select range stops with the character to the left of the cursor.

After selecting a range of text, you can do a variety of operations on the material. You can move, copy, or delete text using ⟨CUT⟩, ⟨PASTE⟩, and ⟨APPEND⟩. Or you can change text using ⟨CHNGCASE⟩, ⟨REPLACE⟩, and ⟨SUBSTITUTE⟩.

Deleting, Copying, and Moving Sections of Text

The CUT function removes the selected range of text from the screen and stores it in the paste buffer. This special buffer holds text during editing. The PASTE function inserts the contents of the paste buffer to the right of the cursor. You will see the text in the select range reappear on the screen.

Practice 4–9

1. Use the EDIT command to get access to the text file you have been using for the practices, or use EDIT/CREATE to start a new file. Type the following text into your file:

   That is the question To be or not to be.

2. Set the select range for the phrase *To be or not to be* by moving the cursor to the *T* in *To* and pressing the ⟨SELECT⟩ key. Advance the cursor until it is at the end of the phrase (one space to the right of the period). Use ⟨CUT⟩ to store the phrase you have selected, in the paste buffer. Move the cursor to the beginning of the sen-

tence (one space to the left of the *T* in *That*). Press ⟨GOLD⟩⟨PASTE⟩ to insert *To be or not to be* at the start of the sentence.

3. When you are familiar with the CUT and PASTE functions use EXIT to store the file.

The only difference between the CUT and APPEND functions is that the CUT function discards any selection previously in the paste buffer and replaces it with the current selection. The APPEND function preserves the selection in the paste buffer and adds the current selection at the end of it.

Changing Case, Replacing, and Substituting Sections of Text

The CHNGCASE function changes uppercase letters to lowercase letters and lowercase letters to uppercase letters in the following cases:

1. With the cursor on a letter, the CHNGCASE function changes the case of only the cursor's character and then, depending on the directional mode, advances or backs up the cursor by one character. However, the CHNGCASE function may have a different effect if you are using the SELECT, FIND, or FINDNEXT functions.
2. If the cursor is on a character within a select range when you press ⟨GOLD⟩⟨CHNGCASE⟩, the function changes the case of each letter in the select range and cancels the select range.
3. When the cursor is on a character that is at a valid search target (and you are not building a select range), the CHNGCASE function changes the case of each character in the search target. The function does not move the cursor.

The REPLACE function erases and discards a select range or search target, inserts the contents of the paste buffer, and places the cursor at the character that follows the insertion. When the paste buffer is empty, the keypad editor erases and discards the selection or search argument and inserts nothing. ⟨GOLD⟩⟨REPLACE⟩ is a convenient way to erase a string that may otherwise require using a combination of delete functions.

The keypad editor always replaces a select range when one exists. The keypad editor replaces a search target only if no select range exists and the cursor is at a valid search target. When you use the REPLACE function to replace a search target, the keypad editor stores the search target in the word buffer.

The SUBSTITUTE function accomplishes both a REPLACE and a FINDNEXT process when the cursor is at a valid search target. (Remember that you can use ⟨FINDNEXT⟩ only after you have used ⟨FIND⟩.) You can change several occurrences of a string to whatever is in the paste buffer by repeating the SUBSTITUTE function. The sequence is:

⟨GOLD⟩ integer⟨GOLD⟩⟨SUBSTITUTE⟩

The keypad editor accepts integers in the range from 1 to 32767.

Practice 4–10

1. Use EDIT to get access to a practice file. Type the following text into your file:

   Although it is Digital's smallest operating system for the pdp–11 processor family, RT–11's features compare very favorably to those of much larger systems.

   Use CHNGCASE to make *pdp–11* uppercase.

   Use REPLACE to erase the string:

   Although it is Digital's smallest operating system for the PDP–11 processor family,.

2. Type the following text into your text file:

   Thank RT–11 parts. How do you choose a material for the structural members of an airplane wing? Or any other members of an aircraft that will travel at great speed? Whatever members are chosen must successfully endure the tremendous stresses of modern flight.

   Use ⟨CUT⟩ to delete the word *parts* from the first line. Then use ⟨FIND⟩ and ⟨SUBSTITUTE⟩ to change each occurrence of *members* to *parts*.
   When you have finished editing your file, use the EXIT command to leave the keypad editor.

Summary

COMMANDS USED WITH KED

EDIT/CREATE	creates a text file
EDIT/INSPECT	allows you to examine a file without making changes to it
EDIT/KED	allows you to modify the contents of a file you created previously
EXIT⟨ENTER⟩	saves the file you created or edited and takes you from the keypad editor mode to the monitor command mode
QUIT⟨ENTER⟩	takes you from the keypad editor mode to the monitor command mode without saving the edits made to a file
SET EDIT KED	makes KED the default keypad editor

FUNCTION KEYS USED WITH KED

⟨←⟩	moves the cursor, character by character, to the right
⟨→⟩	moves the cursor, character by character, to the left

⟨↑⟩	moves the cursor, line by line, upward
⟨↓⟩	moves the cursor, line by line, downward
⟨ADVANCE⟩	sets the direction (forward) in which the cursor will move
⟨BACKUP⟩	sets the direction (backward) in which the cursor will move
⟨BLINE⟩	moves the cursor to the beginning of a line
⟨CHAR⟩	moves the cursor in units of characters
⟨CUT⟩	removes a selected section of text from a file and stores it in the paste buffer
⟨DELCHAR⟩	erases the character at the cursor
⟨DELEOL⟩	erases forward from the cursor to the end of the current line
⟨DELETE⟩	erases the character preceding the cursor
⟨DELLINE⟩	erases forward from the cursor to the beginning of the next line
⟨DELWORD⟩	erases forward from the cursor to the first character of the next word
⟨EOL⟩	moves the cursor to the end of a line
⟨FINDNEXT⟩	searches for the next occurence of the search target, must be used after ⟨GOLD⟩⟨FIND⟩
⟨GOLD⟩	allows you to use the alternate functions of keys on the keypad
⟨GOLD⟩⟨BOTTOM⟩	moves the cursor to the end of a file
⟨GOLD⟩⟨CHNGCASE⟩	changes single letters or letter contents of a select range from uppercase to lowercase or lowercase to uppercase
⟨GOLD⟩⟨FIND⟩	(used with ⟨ADVANCE⟩ or ⟨BACKUP⟩) selects a word or character as a search target, then searches forward or backward for the target
⟨GOLD⟩⟨OPENLINE⟩	inserts a blank line
⟨GOLD⟩⟨PASTE⟩	inserts the contents of the paste buffer to the right of the cursor
⟨GOLD⟩⟨SPECINS⟩	inserts characters in ASCII
⟨GOLD⟩⟨TOP⟩	moves the cursor to the beginning of a file

⟨GOLD⟩⟨UNDELCHAR⟩	restores the last character deleted from the file
⟨GOLD⟩⟨UNDELLINE⟩	restores the last line of text deleted from the file
⟨GOLD⟩⟨UNDELWORD⟩	restores the last word deleted from the file
⟨HELP⟩	displays keypad functions and editor SET functions and explains errors made when a keypad function is used incorrectly
⟨LINEFEED⟩	erases backward from the cursor through the first character of the current or preceding word
⟨REPLACE⟩	erases a selected section of text and inserts the contents of the paste buffer in its place
⟨SUBSTITUTE⟩	accomplishes both the REPLACE and FIND-NEXT functions
⟨WORD⟩	moves the cursor in units of words

References

PDP–11 Keypad Editor User's Guide provides a detailed discussion of KED.

PDP–11 Keypad Editor Reference Card.

5

5 Editing and Issuing Monitor Commands

Monitor commands are commands you type in at your terminal in response to the dot prompt from KMON. You have already used some monitor commands like EDIT, COPY, DIRECTORY, and RENAME. To edit monitor commands, you will use the single line editor (SL). SL is available on Professional 300 terminals and on video display terminals that are compatible with the VT100.

The single line editor allows you to move the cursor within a command line and to delete and replace characters while you are interacting with the monitor. Using SL saves retyping commands in full when you make a mistake, and it is also useful if you are running a program which needs data typed in line-by-line.

This chapter also examines the three ways in which commands may be given to the monitor and discusses when to use each type of command.

When you have completed this chapter, you will be able to start and stop SL; get the SL HELP display; move the cursor to the left and right; delete, restore, and swap characters; and execute all or part of a command line after using SL. You will also learn to cite the difference between commands written in Digital Command Language (DCL), Command String Interpreter (CSI), and Concise Command Language (CCL).

Basics of the Single Line Editor

Much of the time you are using RT–11, you will be issuing monitor commands. If you are typing long or complex monitor command lines you may find that you make typing mistakes. Without using the single line editor (SL) there are only two ways that you can change a line: you can use the ⟨DELETE⟩ key to erase one character at a time to the left of the cursor, or you can use the ⟨CTRL/U⟩ key combination, which erases the entire line, so that you can retype it.

Starting and Stopping

Before you can use SL you must specify your terminal type.

EXAMPLE

If you have a VT102 terminal, the command is:

```
.SET SL VT102⟨RETURN⟩
```

If you are using a VT100 terminal, you need not give this command since VT100 is the default terminal type. You start SL by typing the command, SET SL ON. To stop SL, the command is, SET SL OFF. The commands to specify the terminal type and to start or stop SL can be combined into one command line.

EXAMPLE

```
.SET SL VT102,ON⟨RETURN⟩
```

The ON or OFF part off the command must be the last element in the command line if commands are combined in one line.

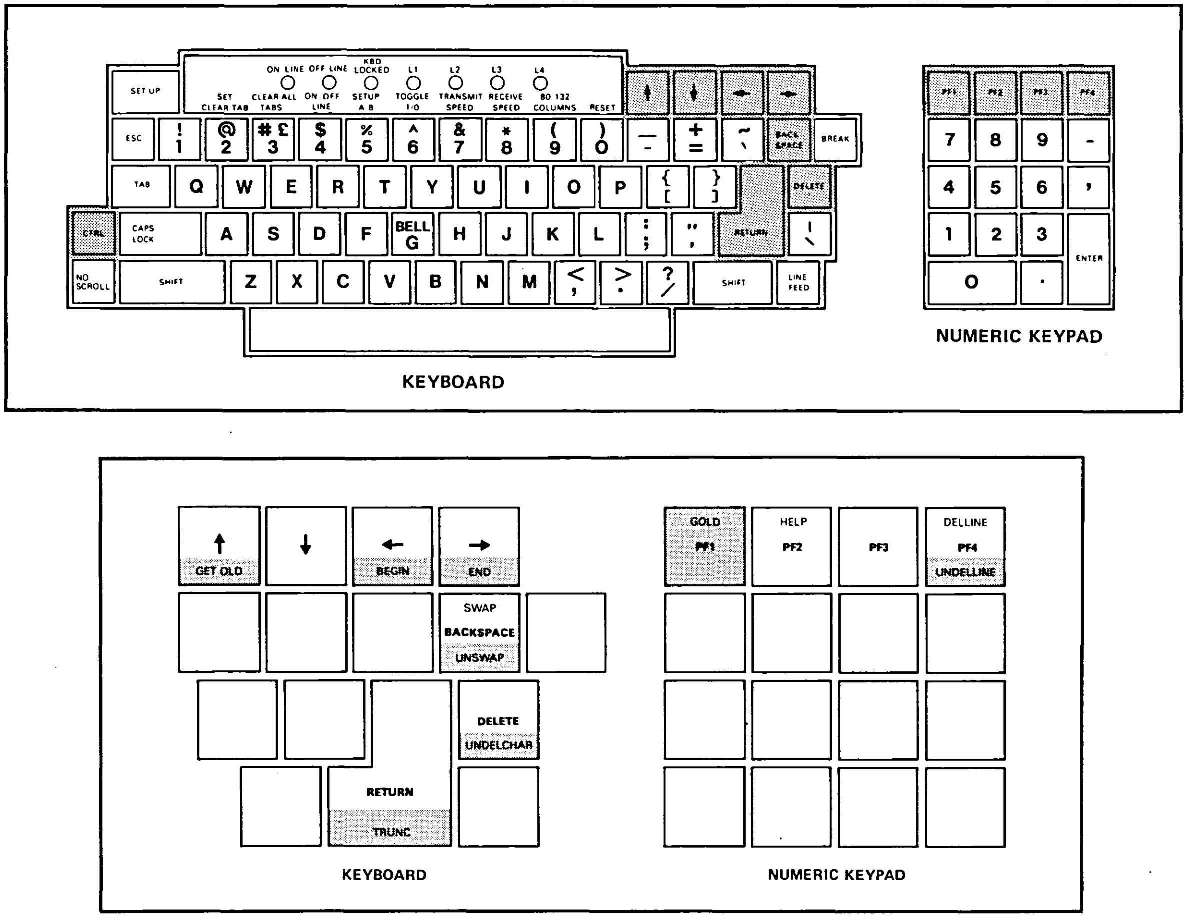


Figure 8. SL Function Keys on the VT100 Terminal

Finding the Functions

When you have issued SET SL ON, you can get a display showing the position of the SL function keys by pressing ⟨HELP⟩. The location of the function keys on the VT100 and Professional 300 keyboards are shown in figures 8 and 9. The keys on the left-hand side of the diagram are on the main keyboard, and the keys at the right of the diagram are on the keypad. The characters and numbers in the center of the keys are the labels that actually appear on the keys; while the terms above and below them indicate the SL functions.

Using SL LEARN to Retain the HELP Display

If you are using a terminal that belongs to the VT100 or Professional 300 family, SL has a special feature to help you become familiar with the position of the keys. When you type the command, SET SL LEARN, and press ⟨HELP⟩, the HELP display will stay on the upper half of your screen even if you type ⟨RETURN⟩ a number of times. The LEARN feature allows you to keep the HELP display on your terminal screen for as long as you need it. To remove the HELP display, give the command, SET SL NOLEARN, which returns the terminal to scrolling normally.

Practice 5–1

1. Type the command:

   ```
   SET SL LEARN⟨RETURN⟩
   ```

2. Press ⟨HELP⟩; then press ⟨RETURN⟩ several times to see what happens to the HELP display.
3. Remove the HELP display with the command:

   ```
   SET SL NOLEARN⟨RETURN⟩
   ```

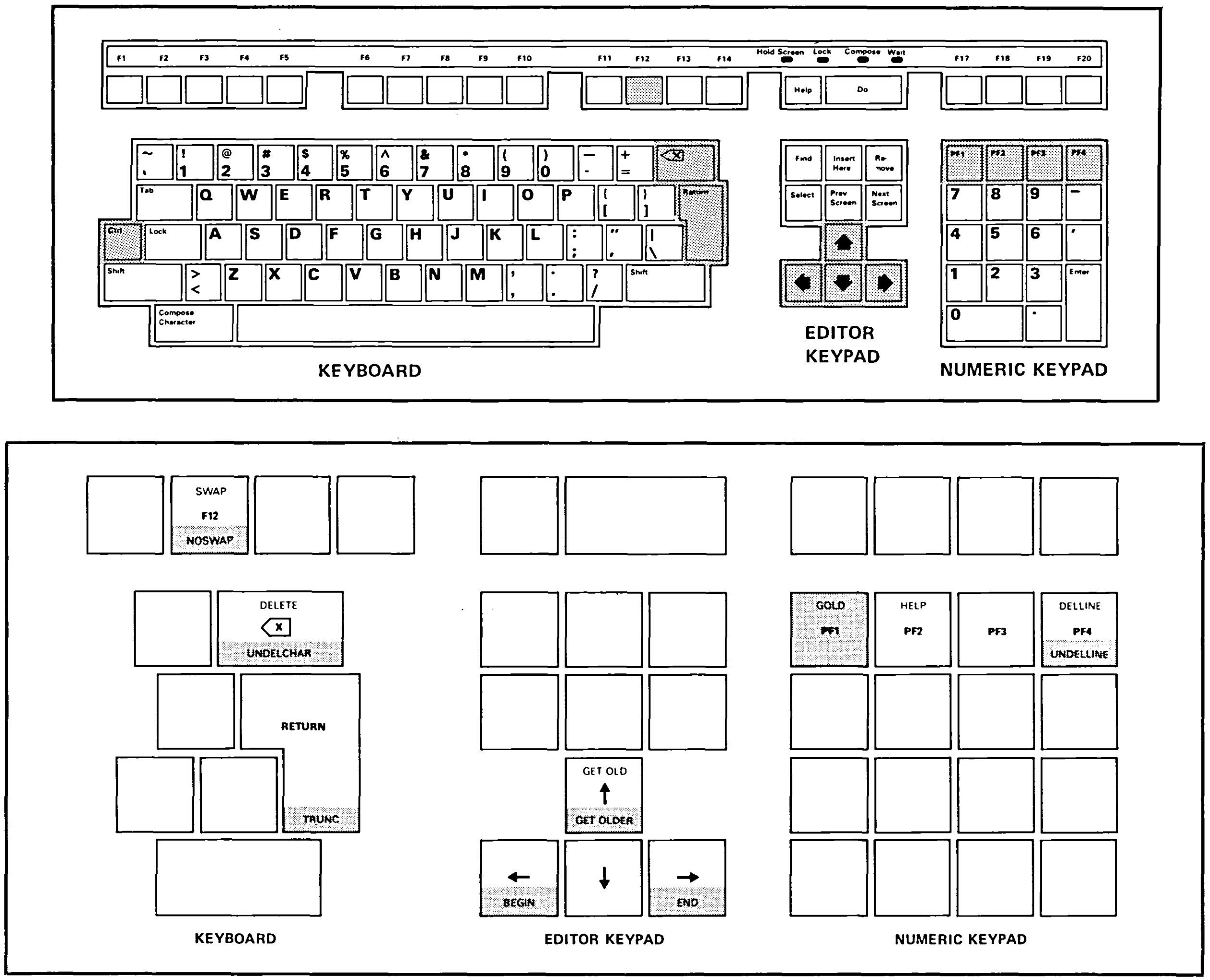


Figure 9. SL Function Keys on the Professional Terminal

The HELP function is also useful if you make an error using SL. When the bell rings after you make an error, you can press ⟨HELP⟩ to get an error message. Press ⟨HELP⟩ again, and you will get the ⟨HELP⟩ display.

To resume editing the material in your file use ⟨CTRL/R⟩. Pressing any key will bring your text back onto the screen, but unless it is ⟨CTRL/R⟩, the character you press will appear in the text.

Using the GOLD Key to Perform Alternate Functions

As with the keypad editor, ⟨GOLD⟩, when used by itself, does not perform a function. However, when used in combination with another function key, it directs SL to perform a specific function. If you press a function key without first pressing ⟨GOLD⟩, the function at the top of the key is performed (see figures 8 and 9). But if you press ⟨GOLD⟩ and immediately press a function key, the alternate function, noted at the bottom of each key, is performed.

Moving the Cursor

You can also use the arrow keys with SL to move the cursor to the left or right one character at a time. To move the cursor to the beginning or end of a line of text you can use the BEGIN or END functions. To get these functions you must press ⟨GOLD⟩ before pressing either ⟨←⟩ or ⟨→⟩.

Performing Operations with the Single Line Editor

Deleting and Restoring Characters

You can delete and restore text by character or by line. ⟨DELETE⟩ erases text one character at a time, starting from the last character typed. All or part of a line of text can be erased by using ⟨CTRL/U⟩ or ⟨DELLINE⟩. ⟨CTRL/U⟩ erases a line of characters to the left of the cursor. ⟨DELLINE⟩ erases a line of characters to the right of the cursor.

If you find you have erased text by mistake, you can restore it in the unit of a character or of a line by pressing ⟨GOLD⟩⟨UNDELCHAR⟩, ⟨GOLD⟩⟨CTRL/U⟩, or ⟨GOLD⟩⟨UNDELLINE⟩.

Exchanging Characters

The SWAP function allows you to switch the positions of the character at the cursor and the character to the right of the cursor. The cursor remains with the same character in its new position. The UNSWAP function switches the positions of the character at the cursor and the character to the left of the cursor. These functions are found on ⟨BACKSPACE⟩ on the VT100 keyboard, and on ⟨F12⟩ on the Professional keyboard.

EXAMPLE

In the examples which follow, the underlined character indicates the position of the cursor.

```
.DIR/PROTECITON⟨SWAP⟩
```

produces:

```
.DIR/PROTECTION
```

Executing a Command Line

When you have completed the changes you want to make in a command line, you can execute the command by pressing ⟨RETURN⟩. You can execute part of the command, from the left margin to the cursor, by using the TRUNC function, that is, by pressing ⟨GOLD⟩ followed by ⟨RETURN⟩.

EXAMPLE

If you have the command string:

```
.COPY DX0:*.MAC DX1:*.BAK
DL1:FILES.BAK
```

and you move the cursor to the D in DL1:

```
.COPY DX0:*.MAC DX1:*.BAK
DL1:FILES.BAK
```

you can execute the command:

```
.COPY DX0:*.MAC DX1:*.BAK
```

by pressing ⟨GOLD⟩⟨TRUNC⟩.

Retrieving Commands

The GET OLD function, obtained by pressing ⟨GOLD⟩ and then ⟨↑⟩ displays the command most recently typed at your terminal. You can then edit the command as if you had just typed it. (If you are using RT–11 version 5.1 on the Professional computer, you simply press ⟨↑⟩ to obtain the GET OLD function while in SL. Pressing ⟨GOLD⟩⟨↑⟩ activates the GET OLDER function, which displays the second-to-last command for you to edit.)

Practice 5–2

1. Enable SL by typing:

   ```
   SET SL ON⟨RETURN⟩
   ```

   Set it to the LEARN option so that the HELP display remains on the screen. Type in the command:

   ```
   DIR/PROTECTION⟨RETURN⟩
   ```

2. Use the GET OLD function to reproduce the command line. Change the command to:

   ```
   DIR/NOPROTECTION⟨RETURN⟩
   ```

by moving the cursor and typing in the extra characters. Execute the command.

3. Use the GET OLD function to redisplay the command line. Delete DIR/NOPROTECTION, with ⟨DELLINE⟩ and type in the following command line exactly as it is shown here:

```
DIR/ALPHABETIZE/RIEBF/CLMNSOU:3/DEELTED
```

Use the SWAP and UNSWAP functions to alter the command line so that it reads:

```
DIR/ALPHABETIZE/BRIEF/COLUMNS:3/DELETED
```

Remember that ⟨GOLD⟩⟨UNSWAP⟩ moves the character at the cursor position to the left, ⟨SWAP⟩ moves the character to the right. Execute the command.

4. Use the GET OLD function to reproduce the command line. Move the cursor to the slash after the command word DIR and use ⟨TRUNC⟩ to execute the first part of the command line and get a complete directory listing.

The REFRESH function—obtained by pressing ⟨CTRL/W⟩ or ⟨CTRL/R⟩—displays the current line of text on your video screen. REFRESH is useful if you are not sure that your screen is displaying information accurately, or if another job prints a message on your screen while you are typing a command line. When you press ⟨CTRL/R⟩ or ⟨CTRL/W⟩, the system removes any interrupting message or data and redisplays the line you are typing.

Simple and Complex Commands

Keyboard monitor commands can be simple or complex. The distinction is based on the way in which the commands are executed rather than on the difference in the format of the command. When you issue a simple keyboard command, it is executed directly by the keyboard monitor, KMON. Simple commands include LOAD, SET, DATE, and TIME.

EXAMPLE

```
.TIME 10:52:03(RETURN)
```

Complex commands are not executed directly by the keyboard monitor. Your RT–11 system has utility programs to perform functions like creating, deleting, or comparing files. The keyboard monitor has to call these utility programs to perform the operations.

When you issue a complex command, KMON automatically expands the command to run a utility program and gives the utility program the file specifications and options.

EXAMPLE

If you issue the command:

```
.COPY INPUT.TMP OUTPUT.TMP(RETURN)
```

KMON will expand it to:

```
.R PIP
*OUTPUT.TMP=INPUT.TMP
```

So, when you use complex commands, you are calling and running utility programs. In order to run a utility program, the system must receive the command, file specifications, and options in a format it can understand. The examples above illustrate simple and complex commands written in the Digital Command Language (DCL) format.

Digital Command Language

The easiest and most convenient command string format to use is Digital Command Language, which works through the keyboard monitor. You do not need to know the name of the utility program being used. Instead, you use a word (one of more than a hundred standard DCL commands) which approximates the name of the task you are trying to accomplish. Once your command has been entered, the system prompts you for file names or other necessary information.

EXAMPLE

```
.COPY(RETURN)
From? OLD.TXT(RETURN)
To   ? NEW.TXT(RETURN)
```

Or you can use the short format:

```
.COPY OLD.TXT NEW.TXT(RETURN)
```

All the monitor commands introduced in this book so far have been DCL commands. The two sections which follow describe other ways to run utility programs.

Commands that Run Utility Programs Directly

Command String Interpreter

The Command String Interpreter (CSI) runs utility programs directly. CSI, known as a parsing utility, is older and less convenient to use than DCL. It consists of typing R (a run command not to be confused with the DCL command, RUN) followed by the name of the utility program.

EXAMPLE

```
.R PIP(RETURN)
```

RT–11 responds to this command to run the peripheral interchange program by giving you an asterisk prompt. You can then type in the file specification in the correct format. When the material in the old file has been copied into the new file, the system sends an asterisk prompt, which means that you are still in the utility program rather than KMON. At this time you can use PIP again simply by typing in the file specification and options you need.

EXAMPLE

```
.R PIP<RETURN>
*NEW.TXT=OLD.TXT<RETURN>
*
```

You press ⟨CTRL/C⟩ to return to KMON when you are finished.

Concise Command Language

Concise Command Language (CCL) also runs utility programs directly and uses the CSI syntax but without the R command.

EXAMPLE

```
.PIP<RETURN>
*NEW.TXT=OLD.TXT<RETURN>
*
```

CCL allows you to use two other formats to simplify the command string:

PIP NEW.TXT=OLD.TXT

or

PIP OLD.TXT NEW.TXT

The second format omits the equal sign and therefore requires a different order in the file specification. In each case, system control returns to KMON when the command has been executed. When issuing CCL and CSI commands, you must supply the name of the utility you are using in each command line.

In addition to DCL, CSI, and CCL, RT–11 provides you with a facility called user command linkage (UCL). UCL is a system generation option that allows you to write your own program (UCL.SAV) to interpret and process additional commands.

Using DCL and CCL to Compare Two Versions of a File

Suppose you want to compare the contents of two files. In Digital Command Language, you would use the DIFFERENCES command.

EXAMPLE

```
.DIF(RETURN)
File 1? US0801.TMP(RETURN)
File 2? US0801.BAK(RETURN)
```

The short command format for this command is:

```
.DIF US0801.TMP US0801.BAK(RETURN)
```

A list of the differences between the files would be printed at your terminal.

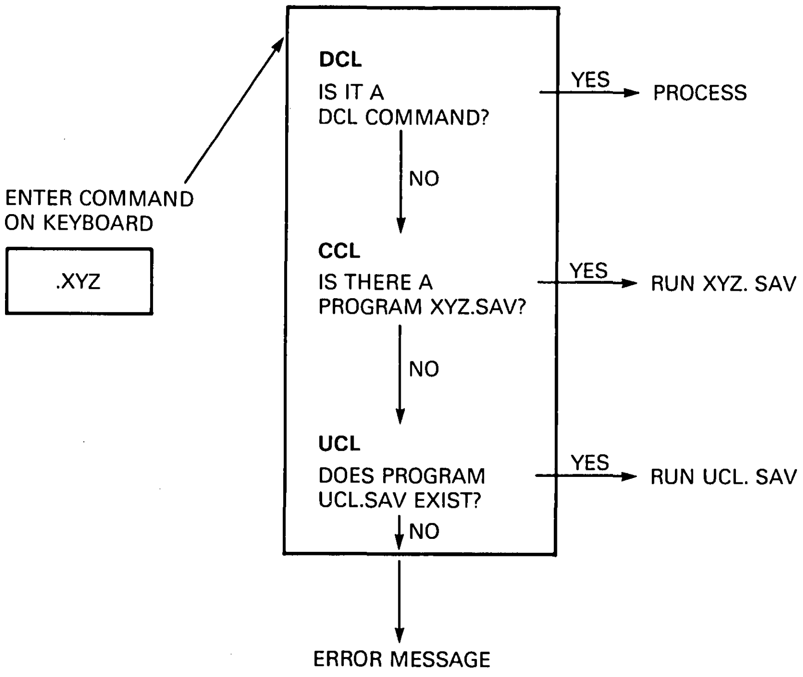
To compare two files using CCL, you must supply the name of the source comparison utility program, SRCCOM.

EXAMPLE

```
.SRCCOM US0801.TMP,US0801.BAK(RETURN)
```

RT–11 runs the SRCCOM utility program regardless of the type of command (DCL or CCL) you use.

Figure 10.
Issuing a Monitor Command



Screening of Monitor Commands

The sequence of events that occurs when you issue a monitor command is shown in figure 10. The keyboard monitor tries the input string against each type of command in turn to check whether or not it is a valid command. As figure 10 shows, KMON goes through a fixed sequence to check a command.

Selecting the Type of Command to Use

You can take advantage of nearly all of the capabilities of RT–11 by using DCL commands. However, you may choose to access and run utility programs directly, using CCL or CSI, on occasion.

If you are a systems programmer and need to use features of the utility programs not accessible from DCL, you would select CCL. If you have used an earlier version of RT–11 or another operating system such as RSTS or OS/8, you may prefer CCL because you are accustomed to running utility programs directly. If you need to use the same utility several times—say, to copy a large number of files—accessing the utility program directly would allow you to type in only the names of the files instead of repeating the entire command.

In general, DCL provides the simplest way to access the utility programs. The long format of DCL is particularly useful because it prompts you for the information needed. For this reason, the next chapter focuses on using Digital Command Language to run utility programs.

Summary

COMMANDS USED WITH THE SL EDITOR

SET SL LEARN	(used with the HELP key) keeps the HELP display on the upper half of the screen
SET SL OFF	stops the single line editor
SET SL ON	starts the single line editor

FUNCTION KEYS USED WITH SL

⟨←⟩	moves the cursor one character to the left
⟨→⟩	moves the cursor one character to the right
⟨CTRL/R⟩	redisplays the current line of text
⟨CTRL/U⟩	deletes the line of characters to the left of the cursor
⟨CTRL/W⟩	redisplays the current line of text
⟨DELETE⟩	deletes each character to the left of the cursor
⟨DELLINE⟩	deletes the characters from the cursor to the end of the line

⟨GET OLD⟩	(on the Professional 300) redisplays the command just typed at the terminal
⟨GOLD⟩⟨BEGIN⟩	moves the cursor to the beginning of line
⟨GOLD⟩⟨CTRL/U⟩	restores the line of characters deleted with ⟨CTRL/U⟩
⟨GOLD⟩⟨END⟩	moves the cursor to the end of a line
⟨GOLD⟩⟨GET OLD⟩	(on the VT100) redisplays the command just typed at the terminal
⟨GOLD⟩⟨GET OLDER⟩	(on the Professional 300) redisplays the second to last command typed at the terminal
⟨GOLD⟩⟨TRUNC⟩	executes the part of a command line to the left of the cursor
⟨GOLD⟩⟨UNDELCHAR⟩	restores the character deleted with ⟨DELETE⟩
⟨GOLD⟩⟨UNDELLINE⟩	restores the line of characters deleted with ⟨DELLINE⟩
⟨GOLD⟩⟨UNSWAP⟩	switches the character at the cursor with the character to the left of the cursor
⟨HELP⟩	displays the location of SL function keys and explains error messages
⟨RETURN⟩	executes an entire command line
⟨SWAP⟩	switches the character at the cursor with the character to the right of the cursor

COMMANDS THAT RUN RT–11 UTILITY PROGRAMS

Command String Interpreter (CSI)
Concise Command Language (CCL)
Digital Command Language (DCL)

References

RT–11 System User's Guide. Section 4 discusses editing command lines with SL.

RT–11 System User's Guide. Chapter 4 lists and explains DCL commands in alphabetical order.

RT–11 System Utilities Manual. Part 1 describes the utility programs available on RT-11.

6

6 Using Utility Programs

The RT–11 system has a wide range of utility programs. You have used a small number of these by entering DCL commands. This chapter will expand your knowledge and allow you to make more efficient use of the system.

The following sections group DCL commands according to the function they perform and describe options which can be used with more than one DCL command. The DCL commands discussed in this chapter are arranged according to these functions: file maintenance, editing and printing, file examination, and system operations.

The relationship between DCL commands and utility programs is complex and therefore one DCL command may run one of several utility programs, depending on the options selected.

When you have completed this chapter, you will be able to use DCL commands to run utility programs that create, copy, delete, rename, or change the protection status of files; get directory listings; print files; or perform system operations such as SET and SHOW.

Utilities and Their Functions

Listed below are some of the functions of utility programs. As you read through, look at the range and types of functions utility programs offer. The groupings of programs are broadly analogous to the groupings of commands you will be studying more closely in this chapter.

File Maintenance

PIP (peripheral interchange program) and FILEX (file exchange) copy files. PIP transfers files from one device to another device on an RT–11 system. It also merges, renames, deletes, and changes the protection status of files. PIP is one of the utility programs you use most often. FILEX converts files from one file format to another so that they can be transferred between, and used with different operating systems. For example, FILEX can create and convert diskettes in IBM format, known as Interchange format, or transfer files from RT–11 to DOS (disk operating system) format.

BUP (backup utility program) copies a file or volume onto a number of smaller volumes for backup storage, and can restore files or volumes from these backup volumes.

DUP (device utility program) is used to create files. DUP is also used to initialize storage media, to detect and cover or replace bad blocks and to copy the bootstrap to a new system volume.

RESORC (resource utility) displays information about your hardware and software configuration at the console terminal.

DIR produces directory listings.

File Editing

In earlier chapters you used one of the RT–11 text editors, KED. KED is really an option for the EDIT command. You call the text editor when you are at monitor level by using the DCL command EDIT/KED.

File Examination

DUMP displays on the console or printer or writes into a file all or part of a file in binary or ASCII.

BINCOM (binary file comparison) and SRCCOM (source comparison) are both used to compare files and list the differences between them. SRCCOM compares text files containing ASCII characters. BINCOM compares files containing data in any binary form other than in ASCII.

Program Development

RT–11 provides facilities to transform a program so that it can be run. These include the linker utility, LINK, and language processors like MACRO–11, DIBOL–11, FORTRAN IV, and BASIC–11.

File Maintenance Commands

In chapter 3, "Storing Data on Disks," you used the following DCL commands: DIRECTORY, COPY, DELETE, RENAME, PROTECT, and UNPROTECT. Along with CREATE, these file maintenance commands are the DCL commands you will probably use most often. In this chapter, you will be concentrating on the various options that allow these DCL commands to perform most efficiently. Options available for only one command are discussed in the section about that command. Except for CREATE, most file maintenance commands use similar options.

The general syntax for a command string with options is:

```
COMMAND/OPTION INFILE/OPTION OUTFILE/OPTION
```

/OPTION represents a qualifier that tells the system either the exact action for a command or more detailed information about a file. Any option you specify after a command applies to the entire command string.

DIRECTORY: Listing the Files on a Storage Volume

The DIRECTORY command, which elicits information about files on a storage volume, has a wide range of options. These options allow you to organize the information displayed, select specific files to be listed, and find volume information. The /ALPHABETIZE, /SORT[:CATEGORY], and /ORDER[:CATEGORY] options allow you to change the order of a listing. /SORT[:CATEGORY] or /ORDER[:CATEGORY] each sorts and displays the directory of a storage device according to a category you specify, for instance, by creation date, file name, file size, or file type. /COLUMNS:n (n can be any integer from 1 through 9) allows you to specify the number of columns in a directory listing. The default value for n is 2.

The /DELETED option lists the files that have been deleted from the device you specify. By supplying a file specification, or simply a device name in the command line, you can get a directory listing showing the details of the file, or the files on the storage device you specified.

The /BLOCKS option includes starting block number in the directory listing. You can use the /FREE option to get a listing of the unused areas and their size.

The DIRECTORY command runs the utility program DIR. You can specify more than one option in the command string.

EXAMPLE

You may want a directory listing that shows the block address of files on DX0: in alphabetic order. The command is:

```
.DIRECTORY/ALPHABETIZE/BLOCKS DX0:<RETURN>
```

COPY: Making Copies of Files

The COPY command which transfers files or the contents of a specified volume from one location to another, has a wide selection of options (see table 11).

Table 11.
COPY Command Options

/ALLOCATE:size	/NEWFILES
/ASCII	/OWNER[:nnn,nnn]
/BEFORE[:date]	/PACKED
/BINARY	/POSITIONS[:n]
/BOOT[:dev]	/PREDELETE
/CONCATENATE	/PROTECTION
/DATE[:date]	/NOPROTECTION
/DELETE	/QUERY
/DEVICE	/NOQUERY
/DOS	/REPLACE
/ENDO:n	/NOREPLACE
/EXCLUDE	/RETAIN
/FILES	/SETDATE[:date]
/IGNORE	/SINCE[:date]
/IMAGE	/SLOWLY
/INFORMATION	/START:n
/INTERCHANGE[:size]	/SYSTEM
/LOG	/TOPS
/NOLOG	/VERIFY
/MULTIVOLUME	/WAIT

EXAMPLE

The DCL command:

```
.COPY/NEWFILES DZ1:*.* DZ0:(RETURN)
```

copies all files with the current date from drive DZ1 to DZ0.

One of three utility programs—PIP, FILEX, or DUP—may be run when the COPY command is issued, depending on the options you specify.

CREATE: Creating or Extending Files

The CREATE command creates or extends a file with a specific name, location, and size. You use the EDIT command with the /CREATE option to start a new text file, but you issue the monitor command CREATE to restore a deleted file. The two are not related.

When you delete a file, the directory entry for the file is deleted. This means that the blocks on the volume can be used for another file, and the contents of the file can be written over. If you use the CREATE command to restore a file as soon as you have deleted it, then the blocks holding the file will not yet have been written over. To create a directory entry for the file you have deleted in error, you will need to specify the starting block number of that file and its size. The CREATE command options /START:n and /ALLOCATE:size allow you to supply this information. You should be cautious when using CREATE to restore a file. The directory may not list the starting block number or size of the most recently deleted file if that file is stored between two files that have been deleted previously.

Another create command option is /EXTENSION:n. Use this option to extend an existing file by the number of blocks you specify. The /EXTENSION:n option follows the file specification.

EXAMPLE

```
.CREATE DX0:TEXT.TMP/EXTENSION:20(RETURN)
```

This command extends the file named TEXT.TMP by 20 blocks.

When you use this option, make sure that there is enough unused space on the volume for the size you specify. (You can use the DIRECTORY/FULL command to check the amount of space available.)

DELETE: Erasing a File

The monitor command DELETE, which erases the files you specify, has several options (two of which are presented here).

/NEWFILES is an option you use to delete files that have the most recent system date. It is a convenient way to remove all the files you just created in a session at the computer.

EXAMPLE

```
.DELETE/NEWFILES DY1:*.BAK<RETURN>
Files deleted:
DY1:TEXT.BAK   ? Y<RETURN>
DY1:MYFILE.BAK ? Y<RETURN>
```

The command issued in the example above told the system to delete all the backup files created today. The command given in the example below tells the system to delete all the files from DY0: except those with the file type .SAV.

EXAMPLE

```
.DELETE/EXCLUDE DY0:*.SAV<RETURN>
?PIP-W-No .SYS action
Files deleted:
DY0:ABC.OLD  ? Y<RETURN>
DY0:AAF.OLD  ? Y<RETURN>
DY0:FILE.OLD ? Y<RETURN>
```

RENAME: Renaming a File

The RENAME command, which assigns a new name to an existing file, has several options. /PROTECTION and /NOPROTECTION allow you to protect or remove the protection status from files using the RENAME command. /REPLACE is the default mode of operation for the RENAME command. If a file exists with the same name as the file you specify for output, the system deletes the duplicate file when it performs the rename operation. /NOREPLACE prevents execution of the rename operation if a file with the same name as the output file you specify already exists on the same device.

PROTECT and UNPROTECT: Preventing Accidental Deletion

The PROTECT and UNPROTECT commands have various options including /BEFORE, /DATE, /NEWFILES, /SETDATE, /SINCE, /EXCLUDE, /QUERY, /SYSTEM, /INFORMATION, /LOG, /NOLOG, and /WAIT. These options are described in the next section.

Common Options for File Maintenance Commands

The options discussed here can be used with more than one DCL command. In particular, they often accompany file maintenance commands. Table 12 lists these options and the commands they qualify.

Table 12.
DCL File Maintenance Commands with Common Options

	Commands				
Options	COPY	DELETE	DIR	PROTECT	RENAME
/BEFORE[:date]	*	*	*	*	*
/DATE[:date]	*	*	*	*	*
/EXCLUDE	*	*	*	*	*
/INFORMATION	*	*		*	*
/LOG	*	*		*	*
/NOLOG	*			*	*
/NEWFILES	*	*	*	*	*
/QUERY	*	*		*	*
/SETDATE[:date]	*			*	*
/SINCE[:date]	*	*	*	*	*
/SYSTEM	*	*		*	*
/WAIT	*	*	*	*	*
/POSITION:n	*	*	*		

Selecting Files By Date

Files are stored with their creation dates in the directory. You can use the creation date of your files to select a group of files to operate on; for example at the end of a working day you may want to copy all the files you have created onto another storage device. A range of options use the creation date as a way of selecting files.

/BEFORE [:date]	selects all files created before a specified date
/DATE [:date]	selects all files with the creation date you specify (if you do not specify a date, then the current system date is used)
/NEWFILES	selects only those files with the current date
/SETDATE [:date]	assigns the date you specify to all files on which the command is performed
/SINCE [:date]	selects all the files created on or after the date you specify

Selecting Specific Files

As you know, if you use wildcards in the file specification, the system allows you to select specific files, or questions you before operating on each file that matches with the wildcard. The following options work similarly.

/EXCLUDE	selects all files on a device except the ones you specify (you can use wildcards in the file specification)
/QUERY	instructs the system to request confirmation before carrying out the command (if you use wildcards in the file specification, this option allows you to check which files are operated on)

/SYSTEM	selects files with the file type .SYS if you have used wildcards in the file specification (unless this option is used, files with the file type .SYS are usually not operated on)

Displaying Information about Selected Files

These options give more information about the files on which the command operates. If you use wildcards in the file specification, you will find these options useful.

/INFORMATION	allows processing of a command to continue after an error is found
/LOG	lists the files on which the command operated (without this option, the system only prints a list of files operated on when there are wildcards in the file specification)
/NOLOG	reverses the /LOG option, so no information about files operated on is displayed.

Applying Commands to Specific Devices

The following options are used for device specific operations.

/WAIT	instructs the system to initiate the command you issued but to pause and wait for you to mount the volume containing the files you want processed (this option is useful if your system has only one disk drive)
/POSITION:n	specifies the block address from which to start searching for a file (this option is for use with operations on magnetic tapes). /POSITION:–1 is used to continue from the current position

Practice 6–1

In this exercise you will use file maintenance commands and some options to manipulate files.

1. Create a text file called FILE.TXT using a text editor.
2. Protect FILE.TXT with the command:

   ```
   PROTECT FILE.TXT<RETURN>
   ```

   You can unprotect a file, rename it, and protect the renamed file in one step with the RENAME command and /PROTECTION option. Type:

   ```
   RENAME/PROTECT FILE.TXT US0901.TXT<RETURN>
   ```

3. Try to delete the file US0901.TXT. You should get an error message like the following:

   ```
   ?PIP-W-Protected file DL0:US0901.TXT
   ```

Commands for Editing and Printing Files

Printing Files on a Line Printer

The PRINT command lets you print files on a line printer. It runs the utility program PIP. You can name up to six files, separated by commas, in a command string. Wildcards can also be used with the PRINT command. PRINT options include /BEFORE, /DATE, /INFORMATION, /LOG, /NOLOG, /NEWFILES, /SINCE, and /WAIT.

Practice 6–2

If you have a line printer, use the PRINT command to produce a hard copy of the file US0901.TXT. Type the command:

```
.PRINT US0901.TXT<RETURN>
```

Typing Files at the Terminal

The TYPE command displays the contents of a text file on the terminal. It also runs the PIP utility program. Use this command when you want to examine a file. Remember that you can use ⟨CTRL/S⟩ and ⟨CTRL/Q⟩ to stop and start the file listing at your terminal and ⟨CTRL/O⟩ to prevent the listing from being displayed. If you are working at a hard copy terminal, you would use the TYPE command to produce a listing of your file.

Practice 6–3 Use the TYPE command to display the contents of US0901.TXT at your terminal.

File Examination Commands

DUMP: Printing Files on the Terminal or Line Printer

The DUMP command allows you to examine directories and files. Output can be printed on the terminal or line printer, or written to a file in octal words, octal bytes, ASCII characters, or Radix–50 characters. The DUMP display will be useful for debugging when you are more familiar with the RT–11 system. Figure 11 shows part of a dump of a text file. The /TER option used here specifies that the dump should be output at the terminal instead of at the line printer, the default output device.

DIFFERENCES: Comparing Files

The DIFFERENCES command lists the differences between files. It may be used when you want to compare two editions of the same file, for example, to see what changes have

Figure 11. Dump of US0901.TXT

```
RK0:US0901.TXT
BLOCK NUMBER  000000
000/ 020040 020040 072101 064440 071564 061440 067157 062543 *    AT ITS CONCE*
020/ 072160 067551 020156 067151 030440 033471 026062 051040 *PTION IN 1972, R*
040/ 026524 030461 073440 071541 062040 071545 063551 062556 *T-11 WAS DESIGNE*
060/ 020144 067564 061040 020145 020141 066563 066141 026154 *D TO BE A SMALL,*
100/ 063040 071541 026164 006440 062412 071541 026571 067564 * FAST, ..EASY-TO*
120/ 072455 062563 067440 062560 060562 064564 063556 071440 *-USE OPERATING S*
140/ 071571 062564 020155 067546 020162 064164 020145 042120 *YSTEM FOR THE PD*
160/ 026520 030461 063040 066541 066151 020171 063157 066440 *P-11 FAMILY OF M*
200/ 067151 061551 066557 072560 062564 071562 006456 044412 *INICOMPUTERS...I*
220/ 020164 060567 020163 062544 062566 067554 062560 020144 *T WAS DEVELOPED *
240/ 071541 060440 071440 067151 066147 020145 071565 071145 *AS A SINGLE USER*
260/ 071440 071571 062564 020155 067546 020162 062562 066141 * SYSTEM FOR REAL*
300/ 072055 066551 020145 067141 020144 067543 070155 072165 *-TIME AND COMPUT*
320/ 072141 067551 060556 006554 072412 062563 020073 072151 *ATIONAL..USE; IT*
340/ 020163 060564 063562 072145 060440 070160 064554 060543 *S TARGET APPLICA*
360/ 064564 067157 020163 062567 062562 062040 072141 020141 *TIONS WERE DATA *
400/ 061541 072561 071551 072151 067551 026156 070040 067562 *ACQUISITION, PRO*
420/ 062543 071563 061440 067157 071164 066157 020054 067141 *CESS CONTROL, AN*
440/ 026144 006440 067412 020146 067543 071165 062563 020054 *D, ..OF COURSE, *
460/ 071160 063557 060562 020155 062544 062566 067554 066560 *PROGRAM DEVELOPM*
500/ 067145 027164 005015 005015 020040 020040 064124 020145 *ENT.....    THE *
520/ 062571 071141 030440 033471 020061 060567 020163 067141 *YEAR 1971 WAS AN*
540/ 062440 061570 072151 067151 020147 064564 062555 063040 * EXCITING TIME F*
560/ 071157 072040 062550 061440 066557 072560 062564 020162 *OR THE COMPUTER *
600/ 067151 072544 072163 074562 020056 064124 020145 042120 *INDUSTRY. THE PD*
620/ 026520 030461 006440 061412 066557 072560 062564 020162 *P-11 ..COMPUTER *
640/ 060567 020163 067157 074554 060440 074440 060545 020162 *WAS ONLY A YEAR *
660/ 066157 020144 067141 020144 044504 044507 040524 020114 *OLD AND DIGITAL *
700/ 060567 020163 060555 064553 063556 061440 066557 072560 *WAS MAKING COMPU*
720/ 062564 020162 067560 062567 020162 062546 071541 061151 *TER POWER FEASIB*
740/ 062554 006440 063012 071157 072040 067550 071565 067141 *LE ..FOR THOUSAN*
760/ 071544 067440 020146 070141 066160 061551 072141 067551 *DS OF APPLICATIO*


BLOCK NUMBER  000001
000/ 071556 073440 072151 020150 064164 020145 067151 071164 *NS WITH THE INTR*
020/ 062157 061565 064564 067157 067440 020146 064164 071551 *ODUCTION OF THIS*
040/ 071040 066145 072141 073151 066145 020171 005015 067151 * RELATIVELY ..IN*
060/ 074145 062560 071556 073151 020145 033061 061055 072151 *EXPENSIVE 16-BIT*
100/ 066440 067151 061551 066557 072560 062564 027162 005015 * MINICOMPUTER...*
120/ 005015 020040 020040 052040 062550 071440 063157 073564 *..     THE SOFTW*
140/ 071141 020145 064164 067145 060440 060566 066151 061141 *ARE THEN AVAILAB*
160/ 062554 063040 071157 072040 062550 050040 050104 030455 *LE FOR THE PDP-1*
200/ 020061 067543 071556 071551 062564 020144 063157 050040 *1 CONSISTED OF P*
220/ 051524 024040 060520 062560 020162 060524 062560 005015 *TS (PAPER TAPE..*
240/ 067523 072146 060567 062562 020054 064167 061551 020150 *SOFTWARE, WHICH *
260/ 067151 066143 062165 062145 072040 062550 050040 046101 *INCLUDED THE PAL*
300/ 030455 051461 040440 071563 066545 066142 071145 020051 *-11S ASSEMBLER) *
320/ 067141 020144 047504 026523 030461 024040 020141 060542 *AND DOS-11 (A BA*
340/ 061564 026550 071157 062551 072156 072141 062145 005015 *TCH-ORIENTATED..*
360/ 074563 072163 066545 027051 041440 062554 071141 074554 *SYSTEM). CLEARLY*
400/ 020054 064164 020145 064563 072564 072141 067551 020156 *, THE SITUATION *
420/ 060543 066154 062145 063040 071157 060440 066040 073557 *CALLED FOR A LOW*
440/ 061455 071557 026164 064440 072156 071145 061541 064564 *-COST, INTERACTI*
460/ 062566 071440 071571 062564 020155 005015 064164 072141 *VE SYSTEM ..THAT*
500/ 061440 072557 062154 061040 020145 071565 062145 063040 * COULD BE USED F*
520/ 071157 071040 060545 026554 064564 062555 060440 062156 *OR REAL-TIME AND*
540/ 061440 066557 072560 060564 064564 067157 066141 060440 * COMPUTATIONAL A*
560/ 070160 064554 060543 064564 067157 026163 060440 062156 *PPLICATIONS, AND*
600/ 063040 071157 006440 070012 067562 071147 066541 062040 * FOR ..PROGRAM D*
620/ 073145 066145 070157 062555 072156 020056 000000 000000 *EVELOPMENT. ....*
```

been made while editing. In the last chapter, you used the DIFFERENCES command to run the SRCCOM utility program.

System Function Commands

SET: Changing Editors, Terminals, or Physical Devices

You may have used the SET command to change the default editor, to change terminal characteristics (SET TT NOSCOPE) or to enable the single line editor (SET SL ON). You can use the SET command to change the characteristics of physical devices, and of system parameters. SET USR NOSWAP, for example, causes the User Service Routines module to remain resident in memory. SET LP LC enables lowercase characters to be printed on a line printer equipped with lowercase type. The SET command is executed by the RT–11 monitor, and does not cause a utility program to run.

SHOW: Getting Information about RT–11

The SHOW command prints information about your RT–11 system on the terminal.

The SHOW command can give you information about your hardware configuration, which version of the monitor you are using, and so on. Options allow you to select details about specific parts of your system.

EXAMPLE

```
.SHOW MEMORY(RETURN)

Address   Module     Words
160000    IOPAGE     4096.
157400    RK          120.
127274    RMON       6170.
126112    DY          313.
```

The ALL option combines the information shown by CONFIGURATION, DEVICES, JOBS, TERMINALS, MEMORY, and SUBSET.

Summary

UTILITY PROGRAMS

BINCOM	(binary file comparison) compares files containing data in any binary form other than ASCII
BUP	(backup utility program) copies a file or volume onto a number of smaller volumes for storage
DUMP	(dump utility) displays all or part of a file in binary or ASCII code
DUP	(device utility program) creates files, initializes storage media, and copies the bootstrap program onto a new system volume
FILEX	(file exchange) converts files from one format to another
PIP	(peripheral interchange program) copies files from one device to another device on an RT–11 system
RESORC	(resource utility) displays information about hardware and software configuration
SRCCOM	(source comparison) compares text files containing ASCII characters

OPTIONS TO FILE MAINTENANCE COMMANDS

COPY/NEWFILES	copies only the files with the most current system date
CREATE/ALLOCATE:size	allocates the number of blocks you specify to the file you are creating

/EXTENSION:n	extends an existing file by the number of blocks you specify
/START:n	specifies the starting block number of the file you are creating
DELETE/EXCLUDE	deletes all the files on a device except the ones you specify
/NEWFILES	deletes only the files with the most current system date
DIRECTORY/ALPHABETIZE	lists the directory of the device you specify in alphabetical order by file name and file type
/BLOCKS	displays a directory of the device you specify and includes the starting block number in decimal of all the files listed
/COLUMNS:n	lists a directory in the number of columns you specify
/DELETED	lists a directory of files that have been deleted from a specific device
/FREE	lists the unused areas on a device
/ORDER[:CATEGORY]	sorts the directory of a device according to the category you specify
/SORT[:CATEGORY]	sorts the directory of a device according to the category you specify
RENAME/NOPROTECTION	renames a file and removes the protected status

/NOREPLACE	prevents execution of the rename operation if a file with the same name as the output file you specify exists on the same device
/PROTECTION	renames a file and gives it protected status so that it cannot be deleted accidentally

OPTIONS COMMON TO FILE MAINTENANCE COMMANDS

/BEFORE[:date]	selects all files created before a specified date
/INFORMATION	allows processing of a command to continue after an error is found
/LOG	lists the files on which the command operated
/NOLOG	reverses the /LOG option, so no information about files operated on is displayed
/POSITION:n	specifies the block address from which to start searching for a file
/QUERY	instructs the system to request confirmation before carrying out the command
/SETDATE[:date]	assigns the date you specify to all files on which the command is performed
/SINCE[:date]	selects all the files created on or after the date you specify
/SYSTEM	selects files with the file type .SYS if you have used wildcards in the file specification
/WAIT	instructs the system to initiate the command you issued but to pause and wait for you to mount the volume containing the files you want to process

DCL COMMANDS

DIFFERENCES	lists the differences between ASCII files (runs SRCCOM)

DUMP	prints a file in octal words, octal bytes, ASCII characters, or Radix–50 characters (runs DUMP)
PRINT	prints files on a line printer (runs PIP)
SET	changes the default editor, terminal characteristics, the characteristics of physical devices, or the characteristics of parameters
SHOW	displays information about your RT–11 system
TYPE	displays the contents of a file on the terminal (runs PIP)

References

RT–11 System Utilities Manual. Part 1 describes all utility programs available on RT–11.

RT–11 System User's Guide. DCL commands and their options are listed alphabetically in chapter 4.

7

7

Developing Programs

As an RT–11 user, you will probably want to solve problems by running applications programs. If you are a programmer, you will write these programs yourself in one of the programming languages supported by RT–11. The process of turning a list of source code instructions (in FORTRAN IV for example) into an error-free working program, is called the program development cycle.

This chapter describes the steps of the program development cycle: creating a source file, compiling or assembling the program, linking object modules to form a complete program, and executing the program.

You will learn to use the appropriate keyboard commands and utilities to assemble or compile the program, link the program, and run the program for a source file in MACRO–11 or FORTRAN IV. You will also learn to identify situations in which you might use program overlays, ODT (the on-line debugging technique), and the librarian utility.

Writing a Program

You have used the system utility programs to perform a wide range of tasks. To solve problems which are specific to your own applications, you may need to write your own programs. RT–11 provides software to help you to do this.

Source Code and Machine Code

Since computers can only understand machine language, programs provided as part of RT–11, such as the utilities, are in machine code (binary) ready to load and run. However, you do not write application programs in machine code but in source code with a programming language such as MACRO–11, FORTRAN IV, or BASIC–11.

EXAMPLE

A typical task for the MACRO–11 programmer is adding the contents of one location in memory to another. The source code statement which adds the contents of Register 1 to Register 0 (a register is a special word in memory) is:

```
ADD R1,R0
```

The equivalent binary machine code for this statement is:

```
0110000001000000
```

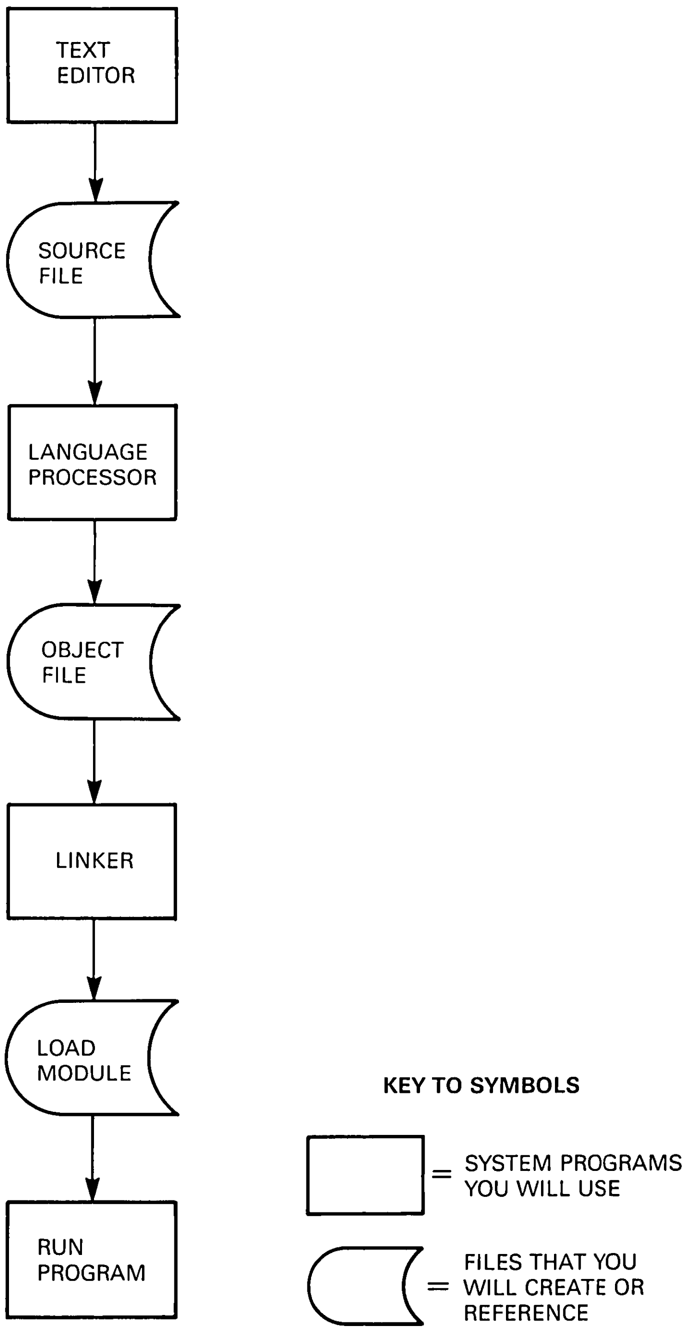
To translate the source code into machine code you will use utility programs provided by the operating system.

The Program Development Cycle

To develop a program you:

1. Use your text editor and a programming language to create a source file.

Figure 12.
The Program Development Cycle



2. Use the appropriate language processor to create an object file which will contain an intermediate form of machine code.
3. Link one or more object files to make a complete working program. The output from the link operation is called a load module.
4. Run the program.

The program development cycle is illustrated in figure 12. We will now look at each step in detail, and by doing the practices you will be able to create a working program from a source file which is provided on your system volume.

Creating the Source File

The source file is the text file which contains the source code of your program. Source files are created using a text editor, such as KED. The file type will usually indicate the language in which the program has been written. For example, PROG.FOR is a FORTRAN IV source file; PROG.MAC is a MACRO–11 source file. A complete list of such file types is given in table 13.

Practice 7–1

Use a text editor to create a source file called US1001.MAC. Type the following text into the file:

```
         .PSECT  US1001
         .MCALL  .TTYOUT,.EXIT,.PRINT,.TTYIN
ASK:     .ASCIZ  /TYPE A NUMBER/
ERMSG:   .ASCIZ  /NON-NUMERIC/

         .EVEN

START:   CLR     R1        ; R1 WILL CONTAIN THE NUMBER INPUT
         .PRINT  #ASK
         JSR     PC,INPUT  ; INPUT A NUMBER INTO R1
         ASH     #1,R1     ; SHIFT 1 POSITION LEFT IE MULTIPLY BY 2
         JSR     PC,OUTPUT ; PRINT THE NUMBER IN R1
         .EXIT
```

```
; READ A NUMBER INTO R1

INPUT:   .TTYIN  R0          ; INPUT A CHARACTER TO R0
         CMPB    #15,R0      ; TEST IF IT'S A <CR>
         BEQ     MULTI       ; END OF NUMBER IF IT IS
         MUL     #10.,R1     ; MULTIPLY NUMBER BY 10 FOR NEXT DIGIT
         SUB     #'0,R0      ; CONVERT NEW CHARACTER FROM ASCII
         CMPB    #9.,R0      ; ERROR IF NEW DIGIT IS GREATER THAN 9 ...
         BLT     ERROR
         TSTB    R0          ; ... OR LESS THAN ZERO
         BLT     ERROR
         ADD     R0,R1       ; ADD NEW DIGIT TO NUMBER SO FAR
         BR      INPUT       ; THEN GO BACK TO READ NEXT CHARACTER
MULTI:   .TTYIN  R0          ; GET <LF>
         RTS     PC

; OUTPUT A NUMBER FROM R1

OUTPUT:  CLR     R0          ; CLEAR HIGH ORDER WORD FOR DIVIDE
         CLR     R2          ; WILL CONTAIN THE CHARACTER COUNT
LOOP:    DIV     #10.,R0     ; DIVIDE BY 10 (REMAINDER IN R1)
         MOV     R1,-(SP)    ; PUSH LEAST SIGNIF. DIGIT ON THE STACK
         MOV     R0,R1       ; SET UP REMAINDER OF NUMBER ...
         CLR     R0          ; ... FOR NEXT DIVIDE
         INC     R2          ; ADD 1 TO CHARACTER COUNT
         TST     R1          ; CHECK IF ANY MORE CHARACTERS
         BNE     LOOP        ; IF THERE ARE, GO BACK FOR THE NEXT
PRINT:   MOV     (SP)+,R0    ; PRINT, GET 1ST (MOST SIGNIF.) DIGIT
         ADD     #'0,R0      ; OFF THE STACK AND CONVERT TO ASCII
         .TTYOUT R0          ; OUTPUT A CHARACTER
         DEC     R2          ; DECREMENT CHARACTER COUNT
         TST     R2          ; TEST FOR MORE CHARACTERS TO DISPLAY
         BNE     PRINT       ; BACK IF MORE
         RTS     PC

; ERROR IF NON-NUMERIC DATA

ERROR:   .PRINT  #ERMSG      ; IF NON-NUMERIC, PRINT ERROR MESSAGE
ERLOOP:  .TTYIN  R0          ; READ NEXT CHARACTER
         CMPB    #12,R0      ; TEST IF <LF>
         BNE     ERLOOP
         .EXIT               ; AND STOP.
         .END    START
```

US1001.MAC contains the source code for a program written in MACRO–11. We will use this program later in the chapter to work through the steps of the program development cycle.

Table 13.
Default File Types

File Type	File Represented
.ANS	SYSGEN answer file
.BAC	Compiled BASIC program
.BAD	Files with unreadable or "bad" blocks; you can assign this file type to defective areas on a device. The .BAD file type makes the file permanent in that area, preventing other files from using it.
.BAK	Backup file created by the text editor
.BAS	BASIC source file (BASIC input)
.BAT	BATCH command file
.BLD	Command file to execute SYSGEN monitor (.MON) and device handler (.DEV) build files
.BUP	Backup utility program output file
.CND	SYSGEN conditional file
.COM	KMON indirect command file, IND indirect control file, or SIPP command file
.CTL	BATCH control file generated by BATCH compiler
.CTT	BATCH internal temporary file
.DAT	BASIC, FORTRAN, or IND data file
.DBL	DIBOL source file
.DDF	DIBOL data file
.DEV	SYSGEN device handler build file
.DIF	BINCOM or SRCCOM differences file
.DIR	Directory listing file
.DMP	DUMP output file
.DSK	Logical disk file (for use with LD handler)
.FOR	FORTRAN IV source file (FORTRAN input)
.LDA	Absolute binary (load image) file (optional linker output)
.LOG	BATCH log file
.LST	Listing file (MACRO, FORTRAN, LIBR, or DIBOL output)
.MAC	MACRO source file (LIBR, MACRO or SRCCOM input)
.MAP	Map file (linker output)
.MLB	MACRO library output file
.MON	SYSGEN monitor build file
.OBJ	Relocatable binary file (MACRO or FORTRAN output, linker input, LIBR input and output)
.REL	Foreground job relocatable image (linker output, default for monitor FRUN and SRUN commands)

Table 13. Continued

File Type	File Represented
.SAV	Memory image (default for R, RUN, SAVE, and GET keyboard monitor commands; default for linker output)
.SLP	SLP command file
.SML	System MACRO library
.SOU	Temporary source file generated by BATCH
.STB	Symbol table file containing the symbols in object format produced during link
.SYG	Monitor and handler files resulting from system generation
.SYS	Monitor files and handlers
.TBL	Monitor device table section created during SYSGEN
.TMP	ERROUT temporary file
.TXT	Text file
.WRK	Temporary work file

Creating the Object File

The first step in translating a source file into machine code is to produce an object file. This is done by special utility programs called language processors, and each programming language has its own language processor.

The object file contains your program in object code, an intermediate code between source code and machine code. Each symbolic instruction is replaced by one or more machine instructions. References to memory locations, for example, names of variables, are replaced by addresses relative to the start of the program or data section. This means that the assignment of absolute addresses in memory is deferred until the program is linked.

Most programs contain references to code or data which is not contained within the source file. For example, all FORTRAN IV programs must include code from the FORTRAN IV object time system (OTS) to perform functions such as opening and closing files. The language processor does not resolve such references; this is not done until the program is linked.

Unless you specify otherwise, the object file created has the same file name as the source file and has the file type OBJ. For example, the object file produced from a FORTRAN IV source file PROG.FOR would be PROG.OBJ.

The type of language processor used depends on the type of programming language you selected. As we mentioned earlier, programming languages can generally be divided into three groups: assembly language, compiled language, and interpreted language.

Assembly Language

In assembly language programs, each line of source code is equivalent to one line of machine code, as you saw in the ADD instruction example. The RT–11 assembly language is MACRO–11.

The language processor which translates an assembly language program into object code is called an assembler. The MACRO–11 assembler is called MACRO, and is invoked by the command:

```
MACRO FILENAME.TYP
```

where FILENAME.TYP is the name of the source file.

EXAMPLE

```
.MACRO PROG.MAC(RETURN)
```

If your source file, like this one, has the default file type MAC, then you can omit the file type from the specification:

```
.MACRO PROG(RETURN)
```

Practice 7–2

US1001.MAC contains the MACRO–11 source code for a program which performs a simple calculation.

1. Use the MACRO–11 assembler to produce an object file for this program. Since it has the default file type of .MAC, you do not have to specify the file type.
2. Use the DIRECTORY command to show all the files which have the name US1001, regardless of their file type. Two files should appear:

   US1001.MAC—the source file

   US1001.OBJ—the object file

Compiled Language

RT–11 supports a number of compiled languages, of which the best known is FORTRAN IV. In FORTRAN, each line of source code produces more than one line of machine code, and the language processor used is called a compiler. The object module produced by the compiler contains object code similar to that produced by the MACRO–11 assembler.

Assuming that you have the FORTRAN IV language processing program on your system, the command to compile your FORTRAN IV program is:

```
FORTRAN FILENAME.TYP
```

where FILENAME.TYP is the name of the FORTRAN IV source file.

EXAMPLE

If your source file is called PROG2.FOR, the command is:

```
.FORTRAN PROG2.FOR(RETURN)
```

The file type can be omitted if the default type is used. The default for FORTRAN IV source files is FOR.

Practice 7–3

To find out whether the FORTRAN IV compiler is available on your system, use the DIRECTORY command to look for the file FORTRA.SAV on your system volume. (On some systems, FORTRAN IV may be contained on a separate volume. If this is the case, you should ask your system manager how to use FORTRAN IV on your system.)

Compilers for other languages, such as COBOL–11, are used in a similar way.

Interpreted Language

Programs written in interpreted languages, such as BASIC–11, do not have the same development cycle as languages such as MACRO–11 and FORTRAN IV. All program development and execution takes place within an interactive program called an interpreter. No object code is produced, because the interpreter translates the source code directly into machine code and executes it a line at a time.

BASIC–11, FORTRAN IV, and COBOL–11 are all high-level languages, that is, programming languages in which the language processor generates more than one line of machine code from each line of source code.

Creating the Load Module

The object file cannot itself be executed, because object code is designed to enable separately compiled program units to be combined and executed as a single program. The file containing this complete program is called the load module.

The load module is produced by a special utility program called the linker, invoked by the keyboard command

LINK. In the simplest instance—when the program consists of one source file and hence one object file—the command to link the program is:

```
LINK FILENAME.TYP
```

where FILENAME.TYP is the name of the object file.

EXAMPLE

If the object code is in a file called EXAMP.OBJ, the command is:

```
.LINK EXAMP(RETURN)
```

Notice that the file type can only be omitted if it is OBJ, the default for object files.

Unless you specify otherwise, the load module which you create takes the file name from the object file and has a file type of SAV. For example, if the object file is called PROG2.OBJ, then the linker will produce a file called PROG2.SAV.

Practice 7–4

1. Use the LINK utility to create the load module for the MACRO–11 program you assembled in practice 7–2. The object file is named US1001.OBJ.
2. Use the DIRECTORY command to list all the files with a file name of US1001. You should now have three files:

   US1001.MAC—the source file (source code)

   US1001.OBJ—the object file (object code)

   US1001.SAV—the load module (machine code)

Object Libraries

It is possible to write a program as a collection of units, or subroutines, which can be compiled or assembled individually. The linker can combine all the separate object files into a single load module. By breaking down programming instructions into units, you may find that some tasks, such as reading data from a file, are common to a number of programs. Such commonly used subroutines can be made accessible to other programs and users by storing the object code in a special file called an object library.

Object libraries are created and maintained by the librarian utility program, LIBR. By giving the appropriate command to the linker, you can include the library subroutines you need into your program. Some object libraries are included with the system, for example the default system subroutine library SYSLIB.OBJ.

Running the Program

The load module produced by the linker is now in executable format (machine code), ready to load and run. The command generally used to run a program is:

```
RUN FILENAME.TYP
```

If you omit the file type, .SAV is assumed.

EXAMPLE

```
.RUN HELLO(RETURN)
```

will tell the system to run the HELLO.SAV program.

As mentioned earlier, you can also run programs by typing simply the program name.

EXAMPLE

If you type:

```
.HELLO<RETURN>
```

CCL (Concise Command Language) looks for a file called HELLO.SAV and, providing it is in executable format, runs the program.

Practice 7–5

The sample program, US1001, used in previous exercises is a MACRO–11 program for performing some simple arithmetic. When you run it, it prompts you for a number, multiplies the number by two, and displays the result.

Use the RUN command to execute US1001. When it asks for a number, type any number between 1 and 9999, and the result will be displayed.

Detecting Errors

In the exercises you have done so far, each step should have produced the expected result, and the program should have run correctly the first time. When you develop and run a program for the first time this seldom happens. Although this book does not teach you how to correct programming errors, the information given in the following sections will help you recognize some of the common problems that can occur during program development.

Command Line Errors

If you make a mistake in the command line, the utility will not be able to produce the correct results. In this case, check all the details, including the file name.

Compilation or Assembly Errors

The language processor checks the source code for syntax errors, such as unmatched parentheses on a line or branches to undefined labels. If it finds any, it displays a warning message. In this case, it is a good idea to run the language processor again with the /LIST option to request a listing showing the lines in which the errors have occurred. When you have identified the errors, use a text editor to correct the source code, and then try the compilation or assembly again.

Link Errors

The linker searches the libraries which you specify and the default system object library, SYSLIB.OBJ, for external references in your program. If there are any unresolved references, a warning is given and the load module may not be produced. The error is usually the omission of an object file or library name in the command line. The problem may also occur if, for example, a subroutine name is misspelled in the source code. In this case, you will need to correct the source file and recompile the program, then link the subroutines again.

Run-time Errors

A program which has been compiled and linked successfully may still not give the expected results when it is run. One of three situations may occur.

1. The program may appear to run and complete correctly, but the results are incorrect.
2. The program may crash, giving a message indicating the cause of the problem. Sometimes this may also cause the operating system to crash. In this case, you will have to reboot RT–11 before you can continue.

3. There is no response from the program after a few minutes. This is usually caused by a loop in the program. To abort the program, press ⟨CTRL/C⟩ twice.

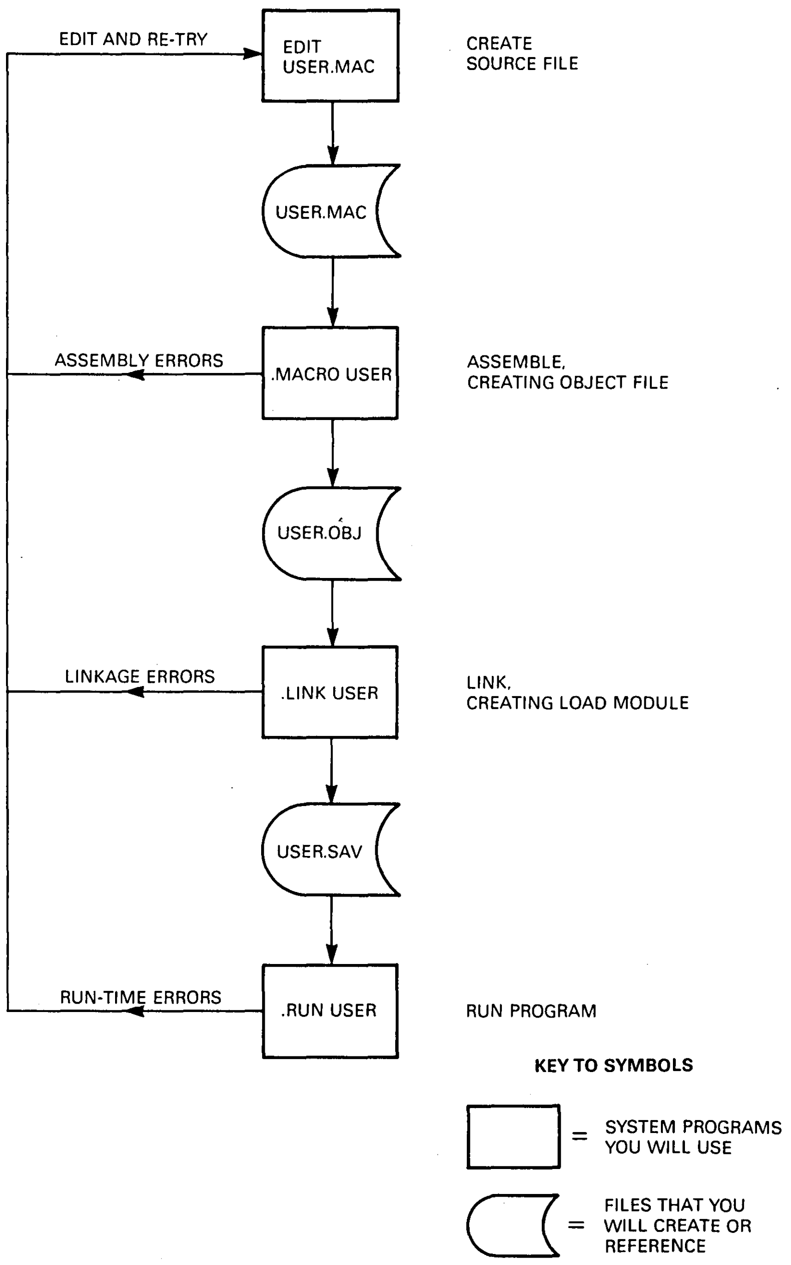
Such errors are usually caused by logic errors in the source code. The process of finding and correcting these errors, or "bugs," is called debugging the program. Sometimes, it is easy to identify the problem by examining a listing of the source code. Another useful technique is to add lines of source code which display the contents of important variables at points before the program fails, and run the program again.

MACRO–11 programmers can make use of a utility called the on-line debugging technique (ODT) to run their programs. This enables you to stop the program at important points and to look at the contents of memory locations. Once you have found the error, make the necessary corrections to the source code using a text editor. You then have to compile, link, and run the program again. Figure 13 shows the program development cycle and indicates the points at which you may have to correct and rerun the program.

Interpretation Errors

The correction cycle is different when you use an interpreted programming language. An interpreter examines each program language statement, interprets it, and executes it before going on to the next. If it discovers an error that prevents further processing, it prints on the terminal a message informing you of the error condition and stops. You correct the error so that execution can continue past that point, and then rerun the program.

Figure 13.
Errors in the Program Development Cycle



Increasing Program Development Efficiency

COMPILE: Combining Several Source Files in an Object File

The COMPILE command calls up the appropriate language processor to assemble or compile several source files into a single object file. The file type of the input source file determines the language processor that is called.

EXAMPLE

```
.COMPILE PROG.MAC(RETURN)
```

tells the MACRO–11 assembler to produce an object file from the MACRO–11 source language statements in PROG.MAC.

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

EXECUTE: Compiling, Linking, and Running with One Command

Like COMPILE, the EXECUTE command instructs a language processor to assemble or compile the source file you specify to produce an object file. It then calls up the linker and runs the resulting load module.

EXAMPLE

```
.EXECUTE MYPROG.FOR(RETURN)
```

generates two files from this FORTRAN IV source file:

MYPROG.OBJ—the object file

MYPROG.SAV—the load module

Program Overlays

In general, the more source code a program contains, the more space it will occupy in memory. On systems without the extended memory option, 28K words of memory is available to a program, although some of this is used by the resident portion of RT–11. On systems which have the extended memory and memory management options, you can write programs that use up to 32K words of memory. If you want to write a program that is too large to fit the memory available to your system, you must use an overlay structure.

In an overlay structure, you write the program in parts so that it can 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 information needed by other segments of the program, called overlay segments.

A program which is to be overlaid must be written in modular form. That is, it should have a main program unit which calls a number of subroutines. When you load a program linked with overlays, the subroutines which are to be overlaid stay on the disk. When one of the subroutines is called, it is loaded to a fixed address within the program space in memory. If another subroutine is then referenced, it is loaded to the same address, overlaying the code for the previous subroutine. This reduces the amount of memory needed to run the program, because not all of the program is loaded at any one time.

EXAMPLE

```
R LINK
!Run the linker program
FILE=FILE, SUBA/F/C
!Build a .SAV with FILE, SUBA,
!FORTRAN OTS library and
SUB1/0:1/C
!Put SUB1 into overlay
!region 1 and continue
SUB2/0:1/C
!Put SUB2 into overlay
!region 1 and continue
SUB3/0:2/C
!Put SUB3 into overlay
!region 2 and continue
SUB4/0:2/C
!Put SUB4 into overlay
!region 2 and end
```

Summary

PROGRAM DEVELOPMENT CYCLE

1. Create a source file (a file which contains your program written in assembly- or a high-level language) using a text editor
2. Assemble or compile the source file into an object file (a file which contains an intermediate form of machine code) using a language processor
3. Create the load module by linking one or more object files into a complete program
4. Run the program

COMMON ERRORS IN PROGRAM DEVELOPMENT

Mistake in a command line
Syntax error in the source code
Omission of the name of an object file or library
Incorrect subroutine name
Error in the logic of the program

DCL COMMANDS USED IN PROGRAM DEVELOPMENT

COMPILE	calls up the appropriate language processor to assemble or compile a source file into an object file
EXECUTE	instructs a language processor to assemble or compile a source file into an object file, calls up the linker, and runs the resulting load module
RUN	runs the load module (the program in machine code) produced by the linker

References

Introduction to RT–11. Chapters 8 through 11, outlines the program development cycle under RT–11.

RT–11 System User's Guide. See the COMPILE and EXECUTE commands in chapter 4. See also section 3.4 for a complete list of standard file types.

Programming with RT–11, Volume 1: Program Development Facilities (Digital Press: Bedford, MA, 1983). This book, the second in The RT–11 Series, provides a thorough introduction to the use of the program development tools outlined in this chapter.

Solutions to Practices

The commands you should type are as follows.

7–1. .EDIT/CREATE US1001.MAC
then type in the source program.

7–2. .MACRO US1001

7–3. .DIR FORTRA.SAV

7–4. .LINK US1001

7–5. .RUN US1001
then type a number between 1 and 9999.

8

8 Creating Files Of Commands

As you work with RT–11, you may find yourself entering a large number of commands in order to complete some tasks. These may be commands to run programs or to organize disk storage. Entering large numbers of commands repeatedly is time consuming, and you may make typing errors. Therefore, RT–11 provides two methods of storing commands for execution at a later time: indirect command files and indirect control files.

This chapter shows you how to create and use indirect command files. To make use of indirect control files, however, you must master a more difficult set of instructions. While we will suggest and illustrate the ways to use these instructions, we cannot hope to make you proficient in the remaining pages of this book.

Understanding Indirect Files

When you type a keyboard command, the keyboard monitor processes it immediately or "directly." If you place the same command in a file, the monitor can process it later or "indirectly" by opening the file and reading it. The file that contains the command is called an indirect file. Such files are useful for storing sequences of commands that you use repeatedly, especially when those sequences require much computer time, but little intervention from you. Examples of such jobs include: assembling multiple source files, compiling programs, making backup copies of disks, or transferring data from one device to another.

RT–11 supports the use of two types of indirect files, indirect command files and indirect control files. Indirect command files contain only monitor commands and any responses that those commands need. Indirect control files contain control instructions in addition to the commands and responses found in command files. You will need to put control instructions in your file whenever you wish to carry out a complex operation that requires periodic attention from you or that makes decisions based on unpredictable information.

Indirect Command Files

You create an indirect command file the same way that you create a text file, by calling up a text editor (such as KED), by naming the file, by supplying a file type, and by typing information into the file—in this case, the commands to be executed.

You can use the EDIT/CREATE command to start a new file. The default file type for indirect command files is .COM. You type each command one to a line and in the sequence it will be executed. You should not type a period (the monitor prompt) before each command. Once you have entered the commands in your file, you can issue the EXIT command to leave the editor and save the newly created file on a disk.

EXAMPLE

To print the time and date and create backup copies of all FORTRAN IV source programs on the default storage device DK: you would type the following commands into a file:

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

When you use wildcards in the file specification of certain commands, like DELETE, the system expects to get a response from you before carrying out the command on each file that fits the specification. If you use such commands in a file and do not include the responses, indirect file processing will stop. There are three things you can do if commands require a response:

1. Use the /NOQUERY option with the command, for example:

   ```
   DELETE/NOQUERY MYFILE.*
   ```

2. Work interactively with the terminal and supply the responses when needed. When this method is used, the process cannot run without you. You inform the system that you want to give the responses at the terminal by placing a special form of ⟨CTRL/C⟩ in the command file, for example:

   ```
   INITIALIZE/NOQUERY/VOLUMEID DM1:
   ^C
   ```

   ⟨CTRL/C⟩ is indicated by a circumflex (^) followed by the letter C. ^C tells the system to take all other input lines from the console terminal. When you finish typing in a response, the indirect command file continues with the next command line.

3. Supply the responses in the command file, for example:

   ```
   INITIALIZE/NOQUERY/VOLUMEID DM1:
   NEWDISK
   DATAPACK
   ```

   Using the /NOQUERY option prevents the need to respond Y[es] to the prompt normally displayed after the INITIALIZE command. The volume identification and owner name needed by the /VOLUMEID option is given in the second and third lines of the file.

You cannot include responses that would destroy data in indirect command files.

EXAMPLE

The following command file:

```
INITIALIZE
DK:
Y
```

does not work because a yes or no response must come from the terminal, not a command file.

You use the /NOQUERY option to prevent the problem.

Adding Comments to Indirect Command Files

You may include comments in indirect command files to help you document your work. These comments do not print on the console terminal when the indirect file executes. Begin each line of comment with an exclamation point (!).

The system ignores anything that it finds between the exclamation point and the end of the current line.

Executing Indirect Command Files

Once you have created an indirect command file, you may start its execution by typing an at sign (@) followed by the name of the indirect command file.

EXAMPLE

```
.@MAKE
```

The file type is assumed to be .COM by default. If you have used any other file type, you must specify it in the command.

EXAMPLE

The following indirect command file assembles and links a MACRO–11 program called MYPROG and then displays on the terminal a directory of all files named MYPROG:

```
! MAKE.COM
!
!This indirect command file assembles
!and links a program called MYPROG.
!The indirect command file is run by
!typing
!
!@MAKE
!
MACRO/LIST/CROSSREFERENCE MYPROG
                       !Do the assembly
LINK/MAP MYPROG        !Link the object
DIR MYPROG.*           !See the files
```

Practice 8–1

Write an indirect command file which displays the current date and time, shows the RT–11 configuration, and lists a directory of all .MAC source files on DK: in alphabetical sequence.

Indirect Control Files

Indirect control files give you more flexibility and control than indirect command files. With indirect control files you can use monitor commands, and you can also use special commands called IND directives to control system execution. IND directives are like keyboard commands but are executed by IND, the indirect control file processing utility, rather than the keyboard monitor.

Each IND directive starts with a period (.). Keyboard commands have no preceding characters when used in a control file. IND processes all commands which start with a period (.) and passes all other commands to the keyboard monitor (KMON). Control files can be used to execute keyboard commands, access files, and perform logical tests to control the flow of execution. Because indirect control files have these features, they resemble programs written in a high-level language. IND directives can be studied as if they were the statements of a programming language.

Creating an Indirect Control File

An indirect control file contains one or more lines of directives or keyboard commands. Each line can contain up to three elements: a label, IND directives or keyboard commands, and a comment.

Labels are used to mark specific locations in a program. IND directives and keyboard commands control the execution of your program and perform specific operations. Comments allow you to document your control program and also to display information. These elements must be arranged in the following format:

```
.LABEL:  IND DIRECTIVE          ;external comment
         or                     or
         KEYBOARD COMMAND       .;internal comment
```

Labels

A label assigns a name to a line so that the line can be referenced. Labels can have up to six alphanumeric or dollar sign ($) characters and must start with a period (.) and end with a colon (:).

EXAMPLE

```
.START:
```

Only one label can appear on each line and it must be placed at the beginning of a line. A label can share a line with directives, keyboard commands, or comments or occupy a line by itself.

Labels allow IND to find a specific sequence of commands in a file. When your control file instructs IND to find a label, IND determines whether or not the label is a direct access label. A direct access label is a label placed on a line by itself; you can define twenty such labels within an indirect control file. (IND maintains a table of up to twenty direct access lables.) If you define more than twenty labels, the new labels replace the old labels beginning with the first one on the table.

When a label is referenced, IND first checks the direct access table. If it is a direct access label, IND goes directly to the label in the file. If it is not, IND searches the file from the cursor to the end, then starts from the beginning of the file and searches to the cursor.

IND Directives and Keyboard Commands

A control line may contain IND directives and monitor commands (either DCL or CCL commands). Each complete command string must fit on one line. Directives and monitor commands can be used together, on the same line or on separate lines. You separate directives from monitor commands with a space or a tab. Some directives can also be used on the same line with other directives.

IND directives allow you to do the following:

- Define labels:

.LOOP:	a name preceded by a period (.) and followed by a colon (:), in this case .LOOP:, can be assigned to a location in your control file, to give you ready access to that location

- Define and assign values to logical, numeric, and string variables. These variables can then be used in place of values in IND directive and monitor command strings:

.SETS LINE "ABC"	sets the string variable LINE equal to ABC
.SETN NUM 10	sets the numeric variable NUM equal to 10

- Create and access data files:

.OPENA #0 COMMND	opens a file so that records can be written to it. The default type is .DAT
.DATA #0 "ABCD"	writes the string ABCD to the file COMMND
.CLOSE #0	closes the file COMMND

- Control the logical flow of processes within a control file:

.GOTO LOOP	go to the line that has the label LOOP

- Perform logical tests:

.IF LINE="ABC" .GOTO LOOP	if the string variable LINE is equal to the literal string "ABC" then go to the line which has the label LOOP

- Enable or disable operating modes:

.ENABLE ESCAPE	when ESCAPE is enabled, IND recognizes ⟨ESC⟩ as a valid response to a question (.ASK, .ASKN, .ASKS)
.ENABLE TIMEOUT	used with questions (.ASKS, .ASKN, .ASK) so that, if a response is not given in a defined period of time, the execution of the control file is aborted. Timeout is available only on systems with timer support and a clock

- Increase or decrease the value of a numeric symbol:

.INC NUM	adds 1 to the numeric variable NUM

- Perform operations which depend on time (if you have timer support):

.DELAY 10S	delays the execution of the control file by 10 seconds
.ASKS[:::20S] NAME WHAT IS YOUR NAME	causes the execution of the control file to be aborted if a response to the question is not given in 20 seconds. The format, nnU (where nn is a number and U is a unit of time), is used to specify the amount of time to wait for a response

Internal and External Comments

Comments can be used for documentation within your file. They can be either internal or external and can be up to 132 characters long—including the period, semicolon, carriage return, and linefeed characters.

Internal comments allow you to document your control file. The comments are not displayed during program execution but are used to explain the internal operations of the control file. Internal comments are defined by a period (.) followed by a semicolon (;).

EXAMPLE

```
.; This is an internal comment
```

They can be used on lines by themselves, on lines with a label, or with a number of directives. You must not use a comment on the same line as a keyboard command, because the keyboard monitor will interpret the comment as an invalid command.

External comments are displayed at the console during program execution to give you information when the control file is run. External comments are defined by a semicolon (;).

EXAMPLE

```
;This is an external comment
```

You can use external comments on lines by themselves, on lines with labels, and with logical test directives. You cannot use them with program control (branching) directives because the branch is performed before the comment is processed. In general, do not use external comments with any directive which displays or contains text (for example .ASK). As a general rule, it is better to put comments on a line by themselves.

The following indirect control file does the same job as the indirect command file that we looked at earlier in the chapter, but allows you to specify an input file name at run-time.

EXAMPLE

```
;get input source file name
.ASKS [1:6] FILE ENTER SOURCE FILE NAME
;add file extension
.SETS FILNAM=FILE+".MAC"
;assemble file here
MACRO/LIST/CROSSREFERENCE 'FILNAM'
;now link the file
LINK 'FILE'
;and take a directory
DIR 'FILE'
```

Executing Indirect Control Files

You can execute indirect control files either from keyboard monitor level or from within another indirect control file (discussed later in the chapter).

To execute an indirect control file from keyboard monitor level, you must call IND.

EXAMPLE

```
.R IND(RETURN)
*
```

The Command String Interpreter (CSI) prints an asterisk at the left margin of the terminal and waits for you to enter the specification of an indirect control file.

To abort execution of a control file, you press (CTRL/C) once if the system is waiting for you to type in data, otherwise you press ⟨CTRL/C⟩ twice. The syntax of the command string is:

```
CONTROL-FILESPEC/OPTION [parameters]
```

CONTROL-FILESPEC represents the control file you want to execute. The default file type is .COM. /OPTION is one or more of the options listed in table 14. Parameters is one or more values (up to nine) that you can pass to the control file. The brackets are not part of the command syntax; they indicate that the parameters are optional. You must separate each parameter you specify with a space.

When SET KMON IND is in effect, you can use the following syntax to execute a control file:

```
@CONTROL-FILESPEC/OPTION [parameters]
```

IND Options

IND options allow you to change the way IND processes and displays a control file. The four IND options (/D, /N, /Q, /T) are listed and explained in table 14.

Passing Parameters

When you give the specification for a control file, you can assign values to nine variables or parameters the system makes available. The parameters are named P1 through P9. Control instructions within a file refer to these parameters for data. The act of assigning values to these parameters is called passing parameters. Passing parameters allows you

Table 14.
IND Options

Option	Function
/D	Deletes the control file when IND has finished processing that file
/N	Directs IND to ignore all keyboard commands in the control file
/Q	Suppresses the display of keyboard commands and their results
/T	Displays each command line that has been processed

to predetermine the value of these variables when you begin execution of a control file. Parameter values can be names of other symbols, numeric values, or character strings.

Nested Indirect Control Files

You can call indirect control files from within an indirect control file. This process is called nesting control files, and is similar to the use of subroutines in programs. You can embed or nest up to three control files within a control file, for a total of four levels of control files.

You use a command line with the following syntax to call a control file from within a control file:

```
@CONTROL-FILESPEC/OPTION [parameters]
```

CONTROL-FILESPEC names the file to which you want to branch. The system assumes that the file type is .COM. /OPTION is one or more of the options listed in table 14. Parameters is one or more values (up to nine) that you want to pass to the control file. You cannot include internal or external comments in this command line.

The .ENABLE GLOBAL directive allows symbols to retain their values on every level of a control file. Without this directive, IND automatically masks all symbols defined by the previous level as it goes from one nested file to another; it recognizes only the symbols defined in the current level of a nested file. When control returns to a previous level, the symbols defined there become available again and the symbols from the lower levels are lost.

Executing Indirect Command Files from Control Files

To call an indirect command file from within an indirect control file, you type a dollar sign and an at sign ($@) before the name of the indirect command file you wish to access. When you pass control to an indirect command file, the keyboard monitor processes and executes the file. Control then returns to the control file from which the indirect command file was called. The format of the command that calls up an indirect command file is:

```
$@FILESPEC
```

The keyboard monitor assumes that any file you specify has the file type .COM.

EXAMPLE

The following command line calls up the indirect command file DYOUT.COM:

```
$@DYOUT<RETURN>
```

IND Directive Summary and Operating Modes

The IND directives are listed in table 15 by category. Use these directives in your control files to direct execution. Table 16 lists the operating modes you can use with .ENABLE and .DISABLE directives. The entry in the scope column refers to whether the operating mode automatically returns to its default setting or remains at its current setting when control passes to a nested control file. Local operating modes return to their default settings; global operating modes keep their current settings.

Table 15.
IND Directive Summary

Directive	Function
Label Definition	
.LABEL:	Assigns a name to a line in the control file so that the line can be referenced
Symbol Definition	
.ASK	Prints a prompt and uses the response to define a logical symbol and to assign the symbol a logical (true or false) value
.ASKN	Prints a prompt and uses the response to define a numeric symbol and to assign it a numeric value
.ASKS	Prints a prompt and uses the response to define a string symbol and to assign it a string value

Table 15. Continued

Directive	Function
.DUMP	Displays local, global, and special symbol definitions
.ERASE	Deletes local or global symbols from the symbol tables
.PARSE	Breaks a string into substrings
.SETD or .SETO	Redefines a numeric symbol to decimal (.SETD) or octal (.SETO) radix
.SETL	Defines a logical symbol and assigns it a logical value
.SETN	Defines a numeric symbol and assigns it a numeric value
.SETS	Defines a string symbol and assigns it a string value
.SETT or .SETF	Defines a logical symbol or redefines bits within a numeric symbol and assigns the symbol or bits a true or false value
.TEST	Tests attributes of a symbol or sting and stores the results in special symbols
.TESTDEVICE	Tests a specified device and stores the device attributes in the special symbol ⟨EXSTRI⟩
.TESTFILE	Determines if a file exists and stores the results in the special symbols ⟨FILSPC⟩ and ⟨FILERR⟩
.VOL	Assigns a volume ID to a string symbol

File Access	
.CHAIN	Closes the current control file, opens another file, and resumes execution
.CLOSE	Closes an output data file
.DATA	Specifies a single line of data to be sent to an output data file
.OPEN	Creates an output data file. If the file you specify with .OPEN already exists, .OPEN creates a new file and will delete the existing file if you subsequently use the .CLOSE directive. Use the .OPEN directive only when you wish to write to a file
.OPENA	Opens an existing file and adds data to it. If the file you specify does not exist, .OPENA creates a new file. Use this directive only when you wish to write to a file

Table 15. Continued

Directive	Function
.OPENR	Opens an existing file for use with the .READ directive. Use this directive only when you wish to read from a file
.PURGE	Discards or closes an output file without making any changes to the file
.READ	Reads the next record from a file into a string variable. The file must have been previously opened with .OPENR
Logical Control	
.BEGIN	Marks the beginning of a begin-end block
.END	Marks the end of a begin-end block
.EXIT	Terminates processing of either the current control file or a begin-end block, and returns control to the previous level; can also assign a value to the numeric symbol ⟨EXSTAT⟩
.GOSUB	Branches to a subroutine within the control file
.GOTO	Branches to another location in the control file
.ONERR	On detecting an error, branches to another location in the control file
.RETURN	Returns control from a subroutine to the line immediately following that subroutine's call
.STOP	Terminates control file processing
Logical Tests	
.IF	Determines whether a symbol satisfies one of several possible conditions
.IFDF or .IFNDF	Determines whether a symbol is defined or not defined
.IFENABLED or .IFDISABLED	Determines whether an operating mode is enabled or disabled
.IFLOA or .IFNLOA	Determines whether or not a device handler has been loaded
.IFT or .IFF	Determines whether a logical symbol is true or false or tests specific bits in a numeric symbol
Execution Control	
.DELAY	Delays control file processing for a specified period of time

Table 15. Continued

Directive	Function
Enable or Disable Operating Modes	
.DISABLE	Disables the operating modes
.ENABLE	Enables the operating modes
Increase or Decrease Numeric Symbols	
.DEC	Subtracts one from the value of a numeric symbol
.INC	Adds one to the value of a numeric symbol

**Table 16.
Operating Modes**

Operating Mode	Default	Scope	Function
DATA	Disabled	Local	When DATA is enabled, IND sends to an output file all lines that follow the .ENABLE DATA directive until a .DISABLE DATA or .CLOSE directive is encountered
DCL	Enabled	Local	When DCL is disabled, IND suppresses execution of keyboard commands in a control file
DELETE	Disabled	Local	When DELETE is enabled, control files are deleted after execution of the file has completed
ESCAPE	Disabled	Global	When ESCAPE is enabled, IND recognizes the escape character as a valid response to an .ASK, .ASKS, or .ASKN directive
GLOBAL	Disabled	Global	When GLOBAL is enabled, symbol names that begin with a dollar sign ($) are recognized as global symbols; that is, these symbols are recognized throughout all levels of control files

Table 16. Continued

Operating Mode	Default	Scope	Function
LOWERCASE	Enabled	Global	When LOWERCASE is enabled, characters typed in response to an .ASKS directive are stored in the string symbol without automatic lowercase to uppercase conversion
MCR	Enabled	Local	When MCR is disabled, IND suppresses execution of keyboard commands in a control file
OCTAL	Enabled	Global	When OCTAL is enabled, the default radix of responses to .ASKN directives and of numeric symbol definitions is octal
PREFIX	Enabled	Global	When PREFIX is disabled, IND suppresses printing of the asterisk (*) before all prompts that result from .ASK, .ASKN, and .ASKS directives, and the semicolon (;) in front of comments
QUIET	Disabled	Local	When QUIET is enabled, IND does not display keyboard command lines
SUBSTITUTION	Enabled	Global	When SUBSTITUTION is enabled, IND replaces symbols with their assigned values
SUFFIX	Enabled	Global	When SUFFIX is disabled, IND suppresses printing of the question mark and [Y/N] notation at the end of an .ASK prompt, and suppresses range, default, timeout, and question type notations for all ASK directive prompts
TIMEOUT	Disabled	Global	When TIMEOUT is enabled, IND recognizes the timeout parameter for ASK directives if your monitor includes timer support
TRACE	Disabled	Local	When TRACE is enabled, IND displays the command line processed

Sample Application

Suppose you want to assemble a program, using the MACRO assembler; to link it, using the LINK program; and finally to run it. You can either:

- Type the keyboard commands at the console, or
- Put the commands in an indirect command file, or
- Put the commands, along with control instructions, in an indirect control file

If you use the first method, you must type the commands each time you want to perform this operation. If you use an indirect command file, you run only the file, but the commands in it cannot be changed. You must use the same file names and carry out the same instructions each time you perform the operation.

Using an indirect control file allows you to get information interactively, check that information, and select options as the system carries out the keyboard commands in the control file.

Before you use the MACRO keyboard command to solve the problem stated above, you must know:

1. The file name and extension of each of the source files you want to assemble. You must have at least one file, and not more than six.
2. Whether or not the files are on the source device.
3. Whether or not an assembly listing is needed, and if so, the name of the listing device or file.
4. Whether a cross-reference listing is needed, and if so, the name of the listing device or file.
5. What defaults will be used if you omit any information.

Figures 14, 15, 16, and 17 show a detailed flow diagram (in four parts) of the problem.

Figure 14.
Control File Flow Diagram (1)

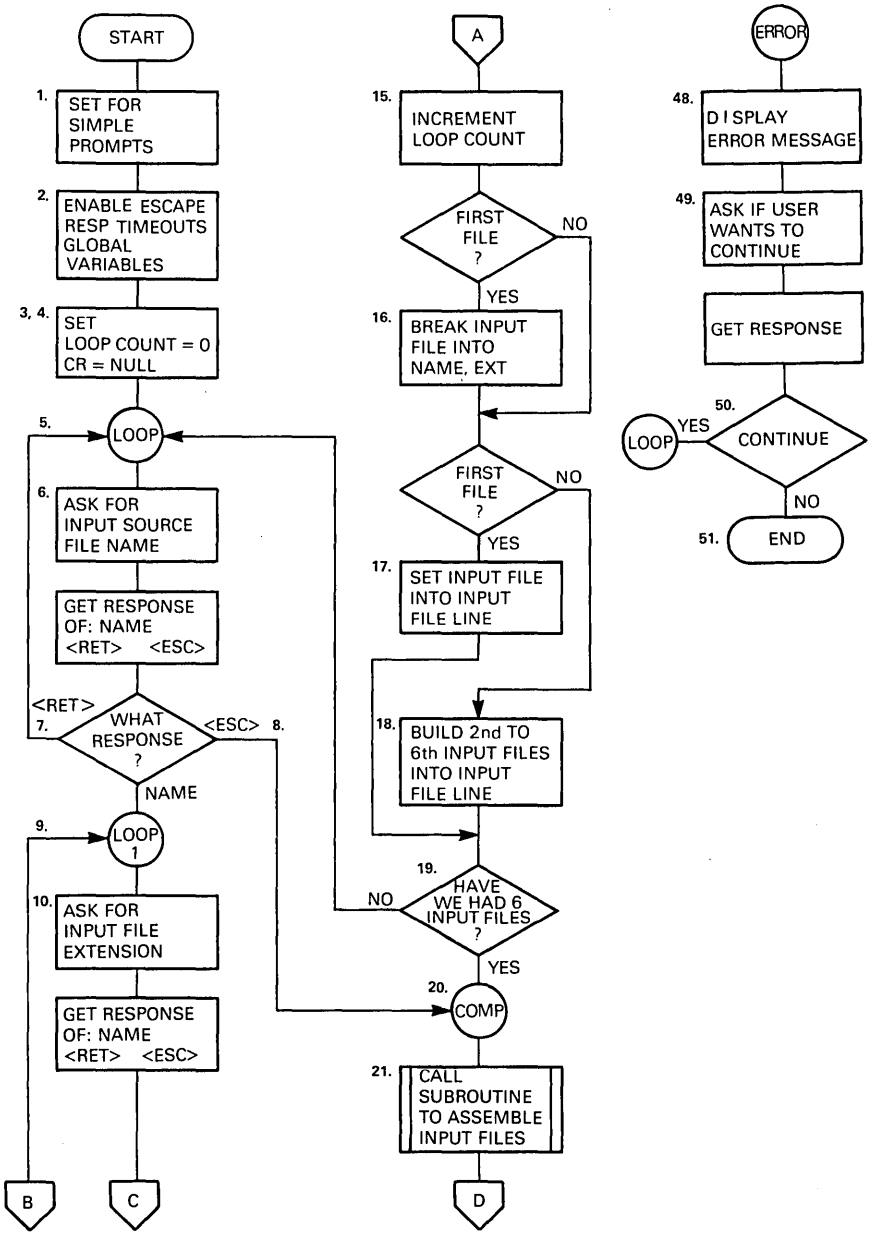

START
1. SET FOR SIMPLE PROMPTS
2. ENABLE ESCAPE RESP TIMEOUTS GLOBAL VARIABLES
3, 4. SET LOOP COUNT = 0 CR = NULL
5. LOOP
6. ASK FOR INPUT SOURCE FILE NAME
GET RESPONSE OF: NAME <RET> <ESC>
7. WHAT RESPONSE ?
<RET>
<ESC> 8.
NAME
9. LOOP 1
10. ASK FOR INPUT FILE EXTENSION
GET RESPONSE OF: NAME <RET> <ESC>
B
C
A
15. INCREMENT LOOP COUNT
FIRST FILE ?
NO
YES
16. BREAK INPUT FILE INTO NAME, EXT
FIRST FILE ?
NO
YES
17. SET INPUT FILE INTO INPUT FILE LINE
18. BUILD 2nd TO 6th INPUT FILES INTO INPUT FILE LINE
19. HAVE WE HAD 6 INPUT FILES ?
NO
YES
20. COMP
21. CALL SUBROUTINE TO ASSEMBLE INPUT FILES
D
ERROR
48. DISPLAY ERROR MESSAGE
49. ASK IF USER WANTS TO CONTINUE
GET RESPONSE
50. CONTINUE
YES
LOOP
NO
51. END

Figure 15.
Control File Flow Diagram (2)

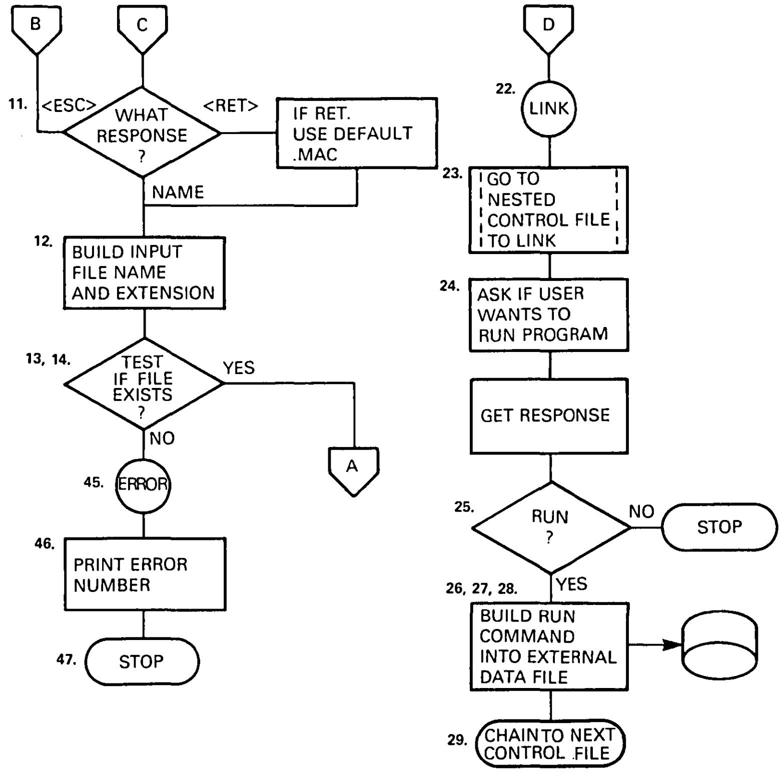


Figure 16.
Control File Subroutine Flow Diagram

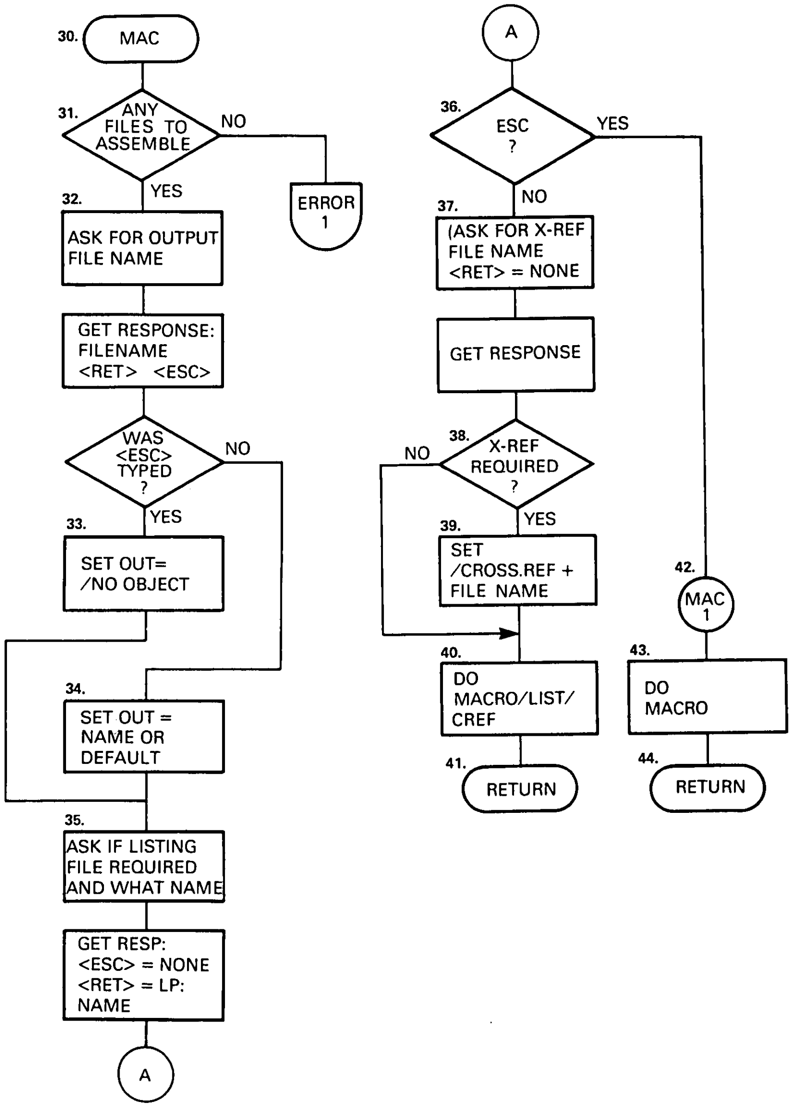

30. MAC
31. ANY FILES TO ASSEMBLE
NO
ERROR 1
YES
32. ASK FOR OUTPUT FILE NAME
GET RESPONSE: FILENAME <RET> <ESC>
WAS <ESC> TYPED ?
NO
YES
33. SET OUT= /NO OBJECT
34. SET OUT = NAME OR DEFAULT
35. ASK IF LISTING FILE REQUIRED AND WHAT NAME
GET RESP: <ESC> = NONE <RET> = LP: NAME
A
A
36. ESC ?
YES
NO
37. (ASK FOR X-REF FILE NAME <RET> = NONE
GET RESPONSE
38. X-REF REQUIRED ?
NO
YES
39. SET /CROSS.REF + FILE NAME
40. DO MACRO/LIST/ CREF
41. RETURN
42. MAC 1
43. DO MACRO
44. RETURN

Figure 17.
Diagram of Nested Control Files

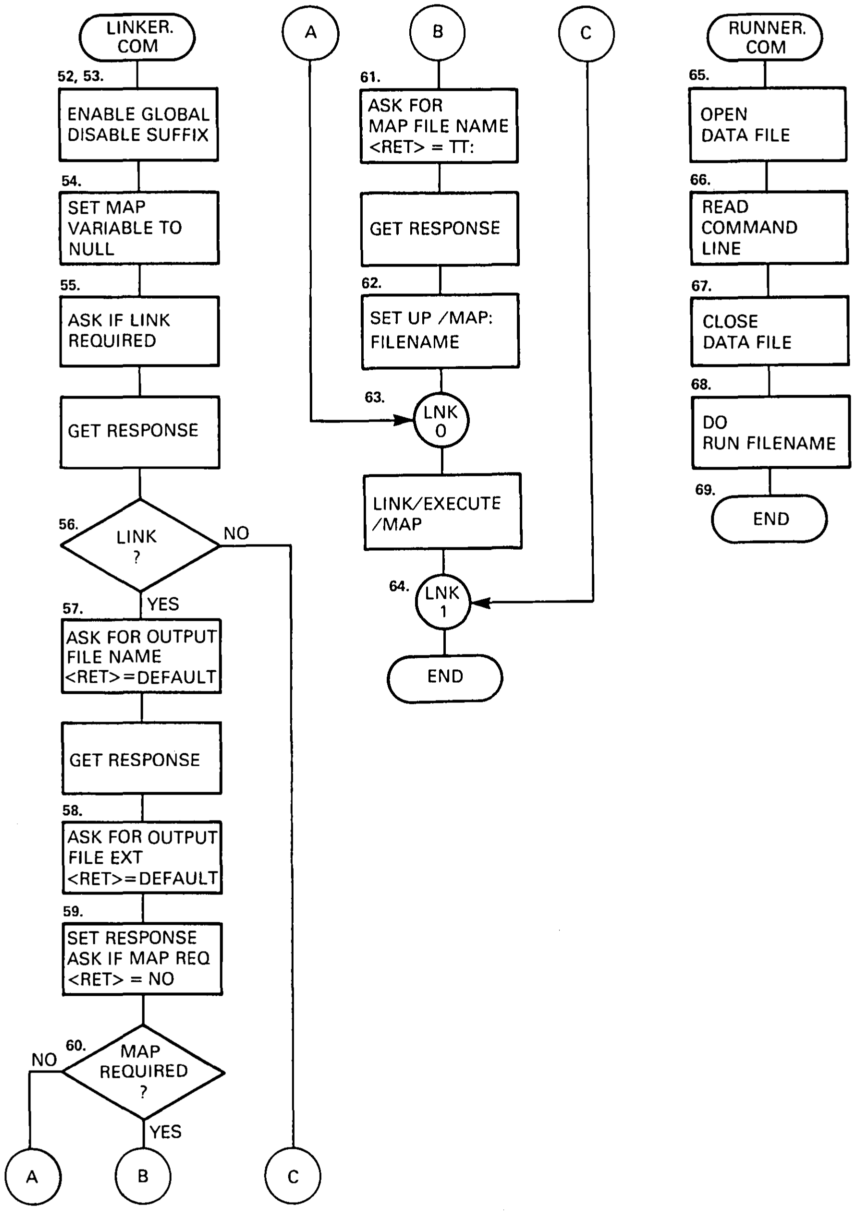


Figure 18. Main Indirect Control File

```
.; US1101.COM          Example Application, Main control file
.;
;This indirect control file obtains the filenames required to assemble
;and link up to six source files into a runnable save  image.  It will
;allow selective listing, cross reference and map files. It is written
;to show a number of IND directives, and could be written more  simply
;if required.
;
       .DISABLE SUFFIX                                                   1
                                  .; remove prompt reminders
       .ENABLE ESCAPE,TIMEOUT,GLOBAL                                     2
                                  .; enable escape and timeout
                                  .; and make $symbols global
       .SETN INCNT 0              .; zeroize loop count                  3
       .SETS CR ""                .; set string variable to null         4
.LOOP:                            .; label on sep line                   5
                                  .; improves efficiency
       .ASKS INFILE IN FILE (type <ESC> if end) ?                        6
                                  .; get in file name
                                  .; <RET> is invalid
                                  .; <ESC> is allowed
       .IFT <ESCAPE> .GOTO COMP   .; if <ESC> go to assemble             7
       .IF INFILE = "" .GOTO LOOP .; if <RET> go back to question        8
.LOOP1:                                                                  9
       .ASKS [::".MAC"] EXT EXT (type <RET> if .MAC)?                   10
                                  .; get extension
                                  .; <ESC> invalid
                                  .; <RET> default
       .IFT <ESCAPE> .GOTO LOOP1  .; if <ESC> return to question        11
       .SETS INFILE INFILE+EXT    .; build filename and extension       12
       .TESTFILE 'INFILE'                                               13
                                  .; check that file exists
       .IF <FILERR> NE 1 .GOTO ERROR  .; if it does not, goto error     14
       .INC INCNT                 .; if success, add 1 to count         15
       .IF INCNT =  1 .PARSE INFILE "." $OUTF $OUTX                     16
                                  .; break filename
                                  .; into 2 strings
                                  .; in global vars
       .IF INCNT =  1 .SETS FILE  INFILE       .; set up 1st file       17
       .IF INCNT <> 1 .SETS FILE  FILE+"+"+INFILE     .; add files 2-6  18
       .IF INCNT LT 6 .GOTO LOOP               .; go for next file      19
.COMP:                                                                  20
       .GOSUB MAC                  .; go to assembly subroutine         21
.LINK:                                                                  22
       @US1102.COM                                                      23
       .; link, using nested control file global vars will be available
       .; ask if user wants to run
       .;
       .ASK [<FALSE>] RN DO YOU WANT TO RUN (type <RET> if no) ?        24
       .;
       .IFF RN .STOP                                                    25
                                  .; if answer <RET> or no, stop
       .OPENA #0 COMMND                                                 26
                                  .; open data file for run command
       .DATA #0 RUN '$OUTF''$EXT' .; write the run command              27
       .CLOSE #0                  .; close the data file                28
       .CHAIN US1103                                                    29
                                  .; and chain out of this
                                  .; control file
```

Figure 18. (Continued)

```
.; ******************** this is a subroutine *************************
.; ******************** it assembles up to 6 source files ************
.; **********as part of main control file, variables are available ****
.MAC:                                                                   30
        .IF INCNT = 0 .GOTO ERROR1          .; cannot assemble without input! 31
                                            .;
                                            .;
        .ASKS [::$OUTF] OUT OUTFILE (type <RET> if def,<ESC> if none)?  32
                                            .;
                                            .; set output file name, default
                                            .; or none, and start testing
                                            .;
        .IFT <ESCAPE> .SETS OUT "/NOOBJECT"      .; set for none if <ESC>   33
        .IFF <ESCAPE> .SETS OUT "/OBJECT:"+OUT        .; set outfile name   34
                                            .;
        .ASKS [::"LP:"] LST LIST FILE (type <ESC>if none,<RET>if LP:)?  35
                                            .;
                                            .; set list file name, LP: or none
                                            .;
        .IFT <ESCAPE> .GOTO MAC1            .; no list, so no x-ref either  36
                                            .;
        .IFF <ESCAPE> .ASK  [<FALSE>] XR X-REF LIST(type<RET>if none)?  37
        .IFT XR .ASKS [::"LP:"] CRLIST X-REF FILE (type <RET> if LP:)?  38
                                            .;
                                            .;
        .IFT XR .SETS CR = "/CROSS-REFERENCE" + CRLIST .; set for x-ref 39
                                            .;
        MACRO/LIST:'LST' 'OUT' 'CR' 'FILE'                              40
                                            .; do the assembly with list
        .RETURN                             .; and return from subroutine 41
.MAC1:                                                                  42
        MACRO 'FILE'                                                    43
                                            .; do the assembly without listing
        .RETURN                             .; and return                44
.; ******************** this is the end of the subroutine *************
                                                                     .;
.; ******************** the error routines start here *****************
.ERROR:                                                                 45
        ;ERROR NUMBER '<FILERR>' OCCURRED                                46
                                            .; note the use of <FILERR>
        .STOP                                                           47
                                            .; stop processing
.ERROR1: ;NO INPUT FILES WERE SPECIFIED                                 48
                                            .; another error
        .ASK [<TRUE>:20.S] EXIT DO YOU WISH TO EXIT?                     49
                                            .; did you mean to
                                            .; finish ?
        .IFF EXIT .GOTO LOOP                .; no, so go round again    50
        .IFT EXIT .STOP                                                 51
                                            .; yes, so stop
```

Figures 18 and 19 show sample control files that were written from the flow diagram. To learn how to use the IND directives and write control files, you should analyze and compare the information in the figures with that of the following section.

Figure 19.
Indirect Control Files

```
.; US1102.COM   Example Application, Linker Control File
.;
        .ENABLE GLOBAL                                              52
        .DISABLE SUFFIX                                             53
        .SETS MP ""                                                 54
        .ASK [<FALSE>] LNK  LINKING (type <RET> if no) ?            55
        .IFF LNK .GOTO LNK1                                         56
        .ASKS [::$OUTF] SAV OUTFILE (type <RET> for default) ?      57
        .ASKS [::".SAV"] $EXT OUT EXT (type <RET> for .SAV) ?       58
        .ASK [<FALSE>] MAP MAP (type <RET> for no)                  59
        .IFF MAP .GOTO LNKO                                         60
        .ASKS [::"TT:"] MAPNAM MAP FILE (type <RET> if TT:)         61
        .IFT MAP .SETS MP "/MAP:"+MAPNAM                            62
.LNKO:  LINK/EXECUTE:'SAV''$EXT' 'MP' '$OUTF'                      63
.LNK1:                                                              64


.; US1103.COM   Example Application, Run Program control file
.;
        .OPENR #0 COMMND.DAT                                        65
        .READ #0 COMLIN                                             66
        .CLOSE #0                                                   67
        'COMLIN'                                                    68
        .STOP                                                       69
```

Analysis of an Indirect Control File

Refer to the text discussions in this chapter and to the flow diagrams as you work through the following analysis of the indirect control file presented in figure 18.

Line	*Directive*	*Discussion*
1	.DISABLE SUFFIX	Prevents display of suffixes. When asking for information, with .ASK, .ASKN, or .ASKS, we can specify default values, timeout periods, and numerical ranges. These items of information are shown in the display as display suffixes.
2	.ENABLE ESCAPE	Enables the use of ⟨ESC⟩ as an operator response. ⟨ESC⟩ is used in the program to indicate that all source file names have been entered.

Line	*Directive*	*Discussion*
2	.ENABLE TIMEOUT	If a response to an .ASK, .ASKN, or .ASKS is not received in a given time, execution will be aborted.
2	.ENABLE GLOBAL	Makes all symbols which start with a dollar sign ($) available to the entire control file, and to any control file which it may call (nested control file).
3,4	.SETN and .SETS	Define or redefine a symbol and assign a value to it. In line 3, for example, a numerical symbol is defined and assigned an initial value of 0. It will be used to control the loop when we process up to 6 input files. The string variable CR will be used later, either as null or to contain the /CROSS-REFERENCE option. In line 4 it is set to null.
5	.label:	Marks a specific line in the control program. Labels may be on the command line or on a separate line. In this case, if the operator presses ⟨RETURN⟩ by accident in line 8, if less than 6 files have been processed in line 19, or if the operator does not press ⟨ESC⟩ in line 6, the program goes back to the label, .LOOP:, in line 5.
6	.ASKS	Displays text and expects a string response. In line six .ASKS is used to get the Macro source file names. Because we enabled ESCAPE in line 2, an ⟨ESC⟩ response at line 6 is allowed and if typed sets a special symbol ESCAPE to true (1). This symbol can later be tested.

Line	*Directive*	*Discussion*
7	.IFT	IF True—one of a number of logical tests. At line 7 it is testing the special symbol ESCAPE to see if ⟨ESC⟩ was typed at line 6. If it was, the program immediately branches to the label .COMP at line 20. This works because ESCAPE was enabled in line 2.
8	.IF	A more general logical test with the format: .IF symbol logical-operator expression There are 6 logical operators: EQ or = NE or <> GE or >= LE or <= GT or > LT or < In this program, we are checking to see if the user pressed ⟨RETURN⟩ by mistake. If ⟨RETURN⟩ has been pressed the program goes back to .LOOP:.
9	.label:	.LOOP: marks the line where we get the source input file extension.
10	.ASKS	With the feature [::".MAC"], allows us to supply a default value .MAC if ⟨RETURN⟩ is typed. The general form of the optional parameters is [low:high:"def":time]. Low and high allow you to specify the character range. You can use values between 0 and 132 decimal. These are the character codes of each character in the user's response. "Def" allows you to specify a default character string. Time allows you to specify a timeout pe-

Line	*Directive*	*Discussion*
		riod in the form nnU where U is either T for ticks (1/10 second), S for seconds, M for minutes, or H for hours. Line 49 shows the time parameter being used. Remember you must enable TIMEOUT (as was done in line 2) before time can be used.
11	.IFT	Tests to make sure that ⟨ESC⟩ was not typed.
12	.SETS	Set String—assigns a value to a string symbol; here it makes one string from the input file name and extension.
13	.TESTFILE	Tests to see if a file is in the directory. In this example, it tests for the specified input file. The results of the test will be placed in a special symbol ⟨FILERR⟩ for later testing.
14	.IF	Tests ⟨FILERR⟩. Success is indicated by a value of 1.
15	.INC	Increments the loop count by 1 if the file is there.
16	.PARSE	Breaks a string into substrings. Here we are separating the file name from the extension and placing them in global variables. The global variables start with $. Globals were enabled in line 2. .PARSE, is implemented only for the first time through the loop.
17,18, 19	.IF	A number of .IF directives that assemble a line for the later MACRO command using .SETS. Line 19 uses a GOTO to return to LOOP.
20	.COMP	Label that serves as entry point for call to subroutine .MAC.

Line	*Directive*	*Discussion*
21	.GOSUB	Calls a subroutine by going to a label which marks the entry point. In this example the label is .MAC: at line 30.
23	@LINKER.COM	Calls a nested indirect control file to do the linking. You may nest to four levels, including the first. Here we pass parameters by using global variables which are available to nested control files.
24	.ASK	Checks whether the user wants to run the new program. Like the .ASKS directive, but this one accepts only Y or N for yes or no. Like .ASKS it uses optional parameters in the form [default:time]. Time is the same as in .ASKS but the default is either ⟨TRUE⟩ or ⟨FALSE⟩, which are special symbols.
25	.IFF	If false—checks what the user typed. If the user typed ⟨RETURN⟩ in response to the previous questions, IFF will use the .STOP directive to stop the program.
26	.OPENA	Opens a file for output to which records will be appended.
27	.DATA	Writes a record to the opened file. In this example it is a keyboard command, RUN, followed by the global variables which contain the file name and extension of the program to run.
28	.CLOSE	Makes the output file permanent and closes the channel to it so that the file can be used to pass commands to KMON.

Line	*Directive*	*Discussion*
29	.CHAIN	Closes this control file and chains to another.

Practice 8–2

Using the analysis above as a guide, finish analyzing the subroutine MAC and the two control files US1102.COM and US1103.COM (lines 30–69 in figures 18 and 19).

Practice 8–3

Modify the file you wrote in practice 8–1 to make it an indirect control file that:

1. Displays the current date and time, then asks if you want to set (or change) either. If you indicate a change, it should ask for the new values and set them for you.
2. Asks which show option you want and uses the answer as an option on the SHOW command.
3. Asks on which device the files are to be listed and which file type you want to see and in what sequence. Does not accept illegal categories for the DIRECTORY/ORDER command and lists the directory as specified.

Summary

ELEMENTS IN INDIRECT COMMAND FILES

Monitor command	is typed one to a line
!	(exclamation point) begins a line of comment

@	(at sign followed by the name of an indirect command file) begins execution of an indirect command file

ELEMENTS IN INDIRECT CONTROL FILES

Label	marks a specific sequence of commands in a file so that IND can easily find it
IND directive	defines labels; assigns values to logical, numeric, and string variables; creates and accesses data files; controls the flow of processes within a file; performs logical tests; enables or disables operating modes; increases or decreases the value of a numeric symbol; or performs operations which depend on time
Monitor command	can be typed on a line with an IND directive or on a line by itself
Internal comment	explains the internal operation of a control file, does not appear on the terminal when the control file is run
External comment	gives you information when the control file is run

COMMANDS THAT EXECUTE CONTROL FILES

From monitor level:
```
.R IND
*CONTROL-FILESPEC/OPTION [parameters]
```
When SET KMON IND is in effect:
```
@CONTROL-FILESPEC/OPTION [parameters]
```
To execute command files within control files:
```
$@COMMAND-FILESPEC
```

References

Introduction to RT–11. Chapter 16 explains how to create and execute an indirect command file.

RT–11 System User's Guide. Chapter 5 explains how to use the indirect control file processor (IND) and describes the IND directives in detail.

Solutions to Practices

8–1 The indirect command file should contain the following commands:

```
DATE
TIME
SHOW CONFIGURATION
DIRECTORY/ALPHABETIZE DK:*.MAC
```

8–3 The indirect control file could contain the following commands:

```
.; US1105.COM
.; SAMPLE SOLUTION TO PRACTICE 8-3
DATE
TIME
   .ASK CHT IS DATE AND TIME CORRECT
   .IFT CHT .GOTO SHOW
   .ASKS DAT ENTER THE NEW DATE (dd-mmm-yy)
DATE 'DAT'
   .ASKS TIM ENTER THE NEW TIME (hh:mm:ss:)
TIME 'TIM'
.SHOW:
   .ASKS OPT ENTER THE SHOW OPTION YOU WANT
SHOW 'OPT'
; You must now specify the files for a directory listing
   .ASKS DEV ENTER THE DEVICE NAME (ddn)
   .ASKS TYP ENTER THE FILE TYPE (typ)
; You can select any of the following options for a
; sorted directed listing.
;
; 1. DATE
; 2. NAME
; 3. POSITION
; 4. SIZE
; 5. TYPE
   .ASKN [1:5:3] SEL ENTER AN OPTION NUMBER
      .IF SEL EQ 1 .SETS CAT "DATE"
      .IF SEL EQ 2 .SETS CAT "NAME"
      .IF SEL EQ 3 .SETS CAT "POSITION"
      .IF SEL EQ 4 .SETS CAT "SIZE"
      .IF SEL EQ 5 .SETS CAT "TYPE"
      DIRECTORY/ORDER:'CAT' 'DEV':*.'TYP'
```

9

9 Conserving Space with Device Support

RT–11 allows you to use a wide range of devices, both physical and logical. This chapter describes device handlers and discusses the use of the virtual memory handler (VM) and the creation of logical disks using the logical disk subsetting utility (LD). You will learn to install and remove a device, use the virtual memory device, and create and mount logical disks.

Using Device Handlers

RT–11 supports a wide range of physical devices most of which have a controller that interfaces the device with the computer. The controllers for some devices, such as disks, support more than one unit or drive. Table 17 shows the devices which are supported by RT–11.

Device controllers need programs called device handlers to drive them. The RT–11 operating system contains a number of device handlers. You will probably not use all

Table 17.
Typical RT-11 Physical Devices

Device	Controller	Device Type
Card reader	CR11	CR11
Clock		KW11–L (line clock) KW11–P (programmable clock)
DECtape II data cartridge	DL11, DLV11	TU58
Disk	RK11, RKV11	RK05, RK05F
	RK611	RK06, RK07
	RL11,RLV11,RLV12	RL01, RL02
	RQDX1	RD51
	UDA–50	RA80
Diskette	RX11, RXV11	RX01
	RXZ11, RXV21	RX02
Display Processor	VT11	
	VS60	
	VS11	
Line Printer	LS11	
	LV11	LV11
	LP11, LPV11	all LP-controlled printers (LP05, LP25, LP26)
Magnetic tape	TM11, TMA11	TU10, TE16
	RH11	TJU16, TU45, TV77
	TS11	TS11, TSV05, TU80
Asynchronous Terminal Interface	DL11, DLV11 DZ11, DZV11 MXV11A, MXV11V	LA120, LA34, LA12 LA100, LQP02, VT100, VT101, VT102, VT105, VT125

of them, but you can see the handlers which are present on your system by issuing the show devices command.

EXAMPLE

```
.SHOW DEVICES(RETURN)
```

The system will then produce a list like the following:

```
Device    Status          CSR       Vector(s)

RK        Not installed   177440    220
DL        Not installed   174400    160
DX        Not installed   177170    264
DY        Not installed   177170    264
VM        Installed       177572    250
LD        Installed       000000    000
DM        Resident        177440    210
LP        Not installed   177514    200
LS        Not installed   176500    310 314
NL        Installed       000000    000
```

The words *Installed* and *Not installed* tell you whether or not the device driver is known to RT–11 (that is, whether or not it is installed in the monitor tables). You install a device by using the following command format:

INSTALL DEVICE[,DEVICE,...DEVICE]

Whenever you add a device that is not part of the original configuration to your system, you must install it or make it known to RT–11. Typically, the person who manages the system is responsible for installing new devices.

EXAMPLE

```
.INSTALL RK:,DY:(RETURN)
```

RK and DY are the permanent device names for the RK05 and RX02 drive units. Thus, the INSTALL command has made the RK05 and RX02 storage devices known to RT–11.

When your system was built, there should have been some empty spaces, or slots, in the device tables. You can find out by typing the SHOW command.

EXAMPLE

```
.SHOW<RETURN>
```

The listing displayed might be as follows:

```
TT (Resident)
DM (Resident)
  DM0=DK, SY
MQ (Resident)
LD
VM
MM
NL
5 free slots
.
```

If there are no available slots, you may remove a device handler from the table by using the following command format:

REMOVE DEVICE:

Using Virtual Memory (VM)

All memory above the 28 Kword boundary is described as extended memory. The Virtual Memory handler (VM) is a device handler that allows you to use all, or part, of extended memory as if it were a disk. Using memory as if it were a disk allows you to make use of high speed memory-to-memory transfers, instead of the slower disk-to-memory transfers.

Transfers between disk and memory take place under the following conditions:

1. Nonresident parts of the operating system are needed.
2. Programs which have overlay sections need an overlay. (As mentioned earlier, an overlaid program is one in which sections of the program replace each other in memory when needed. The process enables you to run long programs with less memory.)
3. Data is needed.

In SJ and FB systems, you can save time by loading the operating system onto the virtual memory device and bootstrapping it from there. You must copy onto the virtual memory device the monitor and the device handlers for the devices you plan to use.

In SJ, FB, and XM you may transfer overlaid programs from the disk into virtual memory and run them. This means that the program will run faster. In these three monitors, data can be worked on as if it were contained in disk files.

Installing the VM Device Handler

To tell RT–11 that you are using the VM device handler, you must install VM in the device handler table. To check whether or not VM has been installed, you enter the SHOW DEVICES command. If VM has not been installed, you can install it with the INSTALL command.

EXAMPLE

```
.INSTALL VM(RETURN)
```

Once you have installed the VM device handler you may use it like any other handler.

Setting the Base Address

Virtual memory must be defined to start at a given memory address. This address is known as the base address. In each of the three types of monitors—SJ, FB, and XM—the virtual memory device handler has a default base address. In SJ and FB, its default base address is 28 Kwords. In the XM monitor, VM's default base address is 128 Kwords. You can change the base address of the VM device handler in each type of monitor by using the SET VM BASE command:

```
SET VM BASE=n
```

The value n used in the command is your selected base address in octal, divided by 100 octal. For example, the address of the 28K boundary is 160000. Divided by 100 octal it becomes 1600. This means that the selected base address must be a multiple of 100 octal, and is therefore fixed at a 32-word boundary.

Before using the SET BASE=n command you must first remove the VM handler, because the amount of memory available to VM is computed at the time the handler is installed. After setting the base address you can reinstall the VM handler. If you change the base address without removing the handler, the system prints a warning message.

EXAMPLE

```
.SET VM BASE=1700(RETURN)

?VM-W-Remove and reinstall this handler
```

You remove the device handler by using the REMOVE command:

```
.REMOVE VM(RETURN)
```

and reinstall it by using the INSTALL command:

```
.INSTALL VM(RETURN)
```

Practice 9–1

The following exercise will acquaint you with the VM handler. If your RT–11 system is equipped with an SJ or FB monitor, VM will be installed by default at a base address of 1600 (i.e, 160000). On a system with the XM monitor, VM will be installed at 10000 (i.e., 1000000) providing that amount of memory exists.

1. Type:

   ```
   SHOW DEVICES(RETURN)
   ```

   to determine if VM is installed.
2. If it is not installed, use the SET command to establish the base address at 1700 and install the VM handler with the INSTALL command.
3. If VM is already installed by default, remove VM, set the base address at 1700 (on an SJ or FB monitor) or at 12000 (on an XM monitor), then install VM.
4. Initialize the VM disk with the INITIALIZE command.
5. To see the amount of storage you have, type:

   ```
   DIR VM(RETURN)
   ```
6. Use the COPY command to transfer a file to VM: and check its directory again.
7. Remove VM, set the base address to 1600 (on an SJ or FB monitor) or 10000 (on an XM monitor) and verify that it is no longer installed.

Using VM in 18-bit Systems

If your system uses 18-bit addressing, you can address up to 124 Kwords of memory. With the SJ and FB monitors, all of the memory between 28 Kwords and the top of memory can be used as virtual memory. If you are using the XM monitor, the memory between 28 Kwords and the top of memory can be shared between an XM program and virtual memory. Figure 20 shows an 18-bit system running the XM monitor in 124 Kwords of memory, with the virtual memory boundary set at 60K words.

Using VM in 22-bit Systems

If your system has 22-bit addressing then you may address up to 2044 Kwords of physical memory. This means that you can use memory from 28 Kwords to the top of memory as virtual memory. With SJ and FB monitors, all of the space is available; with the XM monitor, the space between 28K and 124K is shared with XM programs. If you use the 128 Kword boundary for virtual memory, all of the space that was previously available to XM programs remains available. Figure 21 shows a 22-bit system with the virtual memory boundary set at 128 Kwords.

Using Logical Disks

The RT–11 directory structure allows you to have 72 file entries in each of up to 31 directory segments. This means that you may have a maximum of 2232 (72x31) files on any disk. Because RT–11 supports disks which will hold up to 121 million bytes, you can run out of directory space before running out of file storage space. The problem can be solved with logical disks. Logical disks are files on a physical disk, which are handled as if they were file structured devices.

If your organization uses diskettes, you can create logical disks which have the same capacity as diskettes. You can then copy the logical disk to the physical diskette without space problems or the need to copy files one at a time.

In addition to having more files on each disk, the use of logical disks will permit the creation of sets of files, making applications and program development easier to

Figure 20.
Virtual Memory on an 18-bit System

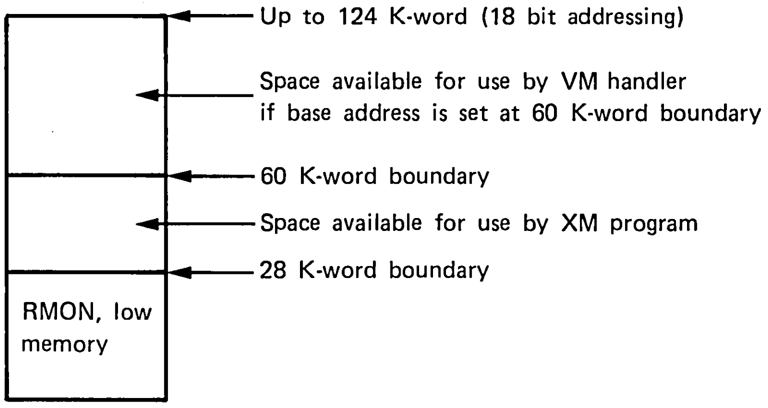
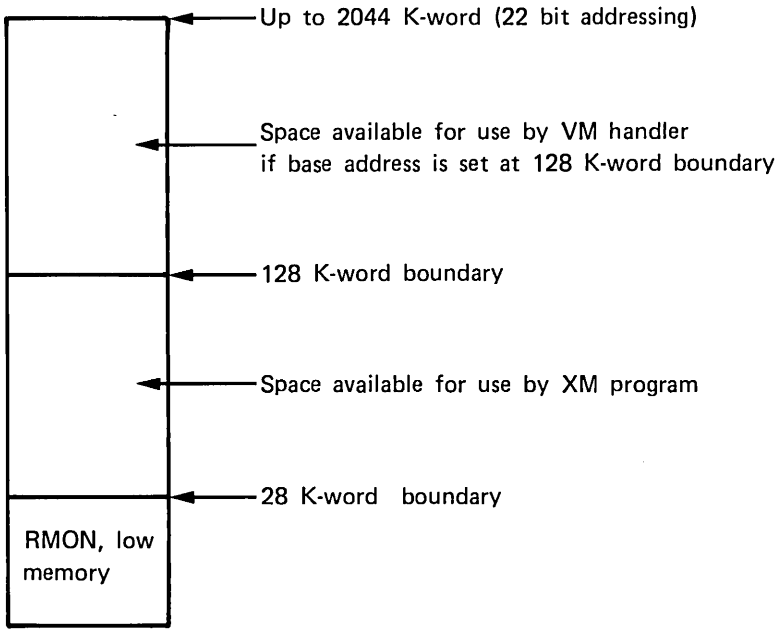

Up to 124 K-word (18 bit addressing)
Space available for use by VM handler if base address is set at 60 K-word boundary
60 K-word boundary
Space available for use by XM program
28 K-word boundary
RMON, low memory

Figure 21.
Virtual Memory on a 22-bit System


Up to 2044 K-word (22 bit addressing)
Space available for use by VM handler if base address is set at 128 K-word boundary
128 K-word boundary
Space available for use by XM program
28 K-word boundary
RMON, low memory

control. For example, if a system has an RK05 disk and two RX01 units you could define logical disks of 494 blocks each on the RK05. Each logical disk could then be used as an RX01 diskette. The logical disks could hold master copies of data for distribution to other systems on RX01, or data from RX01 could be copied to a logical disk on the RK05, which gives faster access time.

To allow you to make use of logical disks, your RT–11 system provides the logical disk subsetting utility (LD). The logical disk subsetting utility allows you to use areas on physical disks as logical disks, each with its own directory structure.

Creating Logical Disks

Before you can define physical disk space as a logical disk, you must create the space you need. Do this by creating a file of the necessary size, with the CREATE command.

EXAMPLE

If you want a logical disk that is 512 Kbytes in size (the capacity of an RX02 diskette) you would give the command:

```
.CREATE DM1:DSKFIL.DSK/ALLOC:1024(RETURN)
```

This command tells the system to create a 1024 block file called DSKFIL.DSK on DM1:. To check that the file has been created you would take a directory listing:

```
.DIR DM1:DSKFIL.DSK(RETURN)
DSKFIL.DSK 1024
1 Files, 1024 Blocks
31025 Free blocks
```

Once you have created the file that the logical disk will need, you can define the logical disk by using DCL commands or by using the logical disk subsetting utility (LD) directly.

The logical disk subsetting utility allows you to perform the following functions:

1. Mount and dismount logical disks and connect them with files on the physical disk.
2. Assign logical names to logical disks.
3. Write-lock a logical disk.
4. Write-enable a logical disk.
5. Verify that the logical disk assignments are correct.

In our discussion, DCL commands are used to run the LD subsetting utility.

The device handler for logical disks is the program LD.SYS, the LD utility program. Make sure that LD is installed by using the SHOW DEVICES command. If necessary, install the device handler with the INSTALL command.

Mounting Logical Disks

The file type .DSK is the default type for logical disk files. When you want to use the file as a logical disk, you can connect it with a logical disk unit number by using the DCL command MOUNT.

EXAMPLE

```
.MOUNT LD0: DM1:DSKFIL.DSK(RETURN)
```

Although we have made the connection between the file and the logical disk number, the logical disk needs a disk structure. We must initialize the disk as if it were a physical disk. The following sequence of commands shows an error message obtained from an invalid command, the INITIALIZE command, and a directory listing.

EXAMPLE

```
.DIR LD0:<RETURN>
?DIR-F-Invalid directory
.INIT LD0:<RETURN>
LD0:/Initialize; Are you sure? Y<RETURN>
.DIR LD0:<RETURN>
0 Files, 0 Blocks
1010 Free blocks
```

Once the logical disk has been initialized, it can be used as if it were a physical disk, (except that you cannot boot from a logical disk). This means that you can also create a logical disk within a logical disk. The following sequence shows a file, LITTLE.DSK, being created within LD0: and turned into LD1: before a data file, SMALL.DAT, is created within it.

EXAMPLE

```
.CREATE LD0:LITTLE.DSK/ALLOCATE:20<RETURN>
.DIR LD0:LITTLE.DSK<RETURN>
LITTLE.DSK 20
1 Files, 20 Blocks
990 Free blocks
.R LD.SYS<RETURN>
*LD0:LITTLE.DSK/L:1<RETURN>
*^C
.DIR LD0:<RETURN>
LITTLE.DSK 20P
1 Files, 20 Blocks
990 Free blocks
.INIT LD1:<RETURN>
LD1:/Initialize; Are you sure? Y<RETURN>
.CREATE LD1:SMALL.DAT/ALLOCATE:10<RETURN>
.DIR LD1:<RETURN>
SMALL.DAT 10
1 Files, 10 Blocks
2 Free blocks
```

Note that the file LITTLE.DSK is shown as protected. The default directory size for logical disks is four segments.

Dismounting Logical Disks

To dismount and disconnect the logical unit number from a file, use the DCL command:

```
DISMOUNT LDn:
```

Assigning Logical Names to Logical Disks

You can assign logical names to logical disks in the same way that you can assign logical names to physical devices. You may make the logical name assignment when you mount the logical disk.

EXAMPLE

```
.MOUNT LD0:DSKFIL.DSK VOL<RETURN>
```

Protecting Logical Disks

You can specify whether or not to write to a logical disk. This is the equivalent of using the write-protect button on a physical disk. When you mount a logical disk, the default selection is for logical disks to be write enabled. However, the MOUNT command takes the option /WRITE or /NOWRITE.

EXAMPLE

```
.MOUNT/NOWRITE LD0: LITTLE.DSK<RETURN>
```

You can also change the setting with the SET command:

```
SET LDn:WRITE
```

or

```
SET LDn:NOWRITE
```

You can use logical disks to increase the number of files stored on a device or to hold a number of related files in one area.

Summary

Logical disk (LD) subsetting utility	allows you to use areas on physical disks as logical disks
Virtual memory (VM) device handler	allows you to use all or part of extended memory as if it were a disk

DCL COMMANDS USED WITH DEVICE HANDLERS

DISMOUNT	disconnects the logical unit number from a file
INITIALIZE	clears and sets up the directory of a logical disk
INSTALL	installs the device you specify on the monitor table
MOUNT	connects the file you want to use as a logical disk with a logical disk unit number
MOUNT/NOWRITE	write-protects the logical disk you are mounting
/WRITE	allows you to write in the logical disk you are mounting (the default operation for MOUNT)
REMOVE	removes a device handler from the monitor table
SET LDn:NOWRITE	write-protects a logical disk

SET LDn:WRITE	allows you to write in a logical disk
SET VM BASE=n	changes the base address of the VM device handler to n (a value you specify)
SHOW DEVICES	lists the handlers that are available on RT–11 and whether or not they have been installed in the monitor table

References

RT–11 System Utilities Manual. Chapter 9 discusses the logical disk subsetting utility (LD).

RT–11 System User's Guide. Chapter 1 includes a brief discussion of device handlers and table 3–1 lists "Permanent Device Names."

RT–11 Software Support Manual. Section 10.12 describes VM.

Suggestions for Reference

PDP–11 Keypad Editor User's Guide
PDP–11 Keypad Editor Reference Card
RT–11 Automatic Installation Booklet
RT–11 Installation Guide
VT100 User Guide

For a complete directory of documentation products, write to:
Digital Equipment Corporation
Circulation Department, MK01/W83
Continental Boulevard
Merrimack, NH 03054

Glossary

Access time The interval between the instant at which data is requested from or for a storage device and the instant at which the data actually begins moving to or from the device.

Address A label, name, or number that designates a location in memory where information is stored.

Alphanumeric The subset of ASCII characters including the 26 alphabetic characters and the 10 numeric characters.

ANSI American National Standards Institute.

Application program A program that performs a function specific to the needs of a particular user or class of users. An application program can be any program that is not part of the basic operating system.

Argument A variable or constant value supplied with a command that controls the commands' action, specifically its location, direction, or range.

ASCII The American Standard Code for Information Interchange; a standard code consisting of eight-bit coded characters for upper- and lower-case letters, numbers, punctuation, and special communication control characters.

Assembler A program that translates symbolic source code into machine instructions. This program replaces symbolic operation codes with binary operation codes and symbolic addresses with absolute or relocatable addresses.

Assembly language A symbolic programming language that can be translated directly into machine language instructions and is specific to a given type of control processing unit.

Assembly listing A listing, produced by an assembler, that shows the symbolic code written by a programmer next to a representation of the actual machine instructions generated.

Asynchronous The type of operation that is triggered by another event, as opposed to synchronous, or occurring at set time intervals.

Background program A program that runs at a low priority, that is, when a higher priority (foreground) program is not using system resources.

Backup file A copy of a file, created as a precaution against loss of the primary file.

BASIC–11 Beginner's All-purpose Symbolic Instruction Code, an interactive, algebraic programming language that combines English words and decimal numbers. This standardized, simple language can handle industrial and business applications.

Binary The number system with a base of two; used by the internal logic of all digital computers.

Binary code A code that uses two distinct characters, usually the numbers 0 and 1.

Bit A binary digit. The smallest unit of information in a binary system of notation. It corresponds to a 1 or 0 and to one digit position in a physical memory word.

Block A group of physically adjacent words or bytes of a size that is specific to a device. For input/output operations, the smallest addressable unit on a mass storage device.

Bootstrap A technique or routine whose first instructions are sufficient to start a system of programs that bring an operating system into memory.

BOT Beginning Of Tape, a reflective marker that is applied to the backside of magnetic tape and identifies the beginning of the magnetic tape'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, often a special register or a designated area of memory, used to hold information being transferred between two devices or between a device or memory.

Bug A flaw in the design or implementation of a program; a problem that can cause erroneous results.

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.

Character A single letter, numeral, or symbol used to represent information.

Clock A device within a computer system that keeps time, counts pulses, measures frequency, or generates regular periodic signals for synchronization.

Code A system of symbols used to represent data or instructions that are executed by a computer.

Coding The writing of instructions for a computer, using a system of symbols that is meaningful to a computer, an assembler, a compiler, or a language processor.

Command A word, mnemonic, or character that, by virtue of its syntax in an input line, causes a computer system to perform a predefined operation.

Command language The vocabulary used by a program or set of programs that directs the computer system to perform predefined operations.

Command language interpreter The program that translates a predefined set of commands into instructions that a computer system can interpret.

Command string A line of input entered into a computer system that generally includes a command, one or more file specifications, and optional qualifiers.

Compile To produce binary code from the symbolic instructions of a high-level source language.

Compiler A program that translates a high-level source language into machine instructions.

Computer A machine that can be programmed to execute a set of instructions.

Computer program A plan or routine for solving a problem on a computer.

Computer system A data processing system that consists of hardware devices, software programs, and documentation that describes the operation of the system.

Concatenation The joining of two or more strings of characters to produce a single string.

Cursor A visible reference point on the display screen which shows where the next entry is to be made.

Configuration A selection of hardware devices, software routines, or programs that function together.

Console terminal A keyboard terminal that acts as the primary interface between the computer operator and the computer system. The console terminal is used to initiate direct system operations by running software on the computer.

Constant A value that remains the same throughout a distinct operation (compare with variable).

CPU Central Processing Unit, a hardware unit of a computer that includes main memory and the registers and circuits that control the interpretation and execution of instructions.

Crash A hardware crash is the failure of a particular device to operate; the operation of an entire computer system may be affected. A software crash is the result of an operating system malfunctioning; the system's protection mechanisms may have failed or the software may not have executed correctly.

Create To open, write data to, and close a file for the first time.

Cross-reference listing A printed listing that links references in a program to specific symbols in a program. It also lists and defines all the symbols used in a source program.

Data A term used to denote information (in the form of numbers, letters, and symbols) that can be processed by a computer.

Data base An organized collection of interrelated data items that allow one or more applications to process the items, while disregarding physical storage locations.

Data collection To bring data from one or more locations to a central location for eventual processing.

Debug To detect, locate, and correct coding or logic errors in a computer program.

Default The value of an argument, operand, or field assumed by a program if a value is not supplied by the user.

Define To assign a value to a variable or constant.

Delimiter A character that separates, terminates, or organizes elements of a character string, statement, or program.

Device A hardware unit such as an I/O peripheral, magnetic tape drive, or line printer.

Device control unit A hardware unit that electronically supervises one or more of the same type of devices. It acts as the link between the computer and the I/O devices.

Device handler A routine that services and controls the hardware activities of an I/O device.

Device name A unique name that identifies each device unit on a system. It consists of a two-letter device mnemonic followed by an optional device unit number and a colon. For example, the common device name for the RL02 disk drive unit 1 is DL1:.

Device unit One of a set of similar peripheral devices, an example of a device unit is disk unit 0.

Digit A character used to represent one of the nonnegative integers smaller than the radix (for example, in decimal notation, one of the characters 0 to 9; in octal notation, one of the characters 0 to 7; in binary notation, one of the characters 0 and 1).

Direct access See Random access.

Directive Assembler directives are mnemonics in an assembly language source program that are recognized by the assembler as commands to control a specific assembly process.

Directory A file in the form of a table containing the names of and pointers to files on a mass storage volume.

Directory-structured A storage volume is directory structured if the directory at the beginning of the volume contains information (file name, file type, length, and date-of-creation) about all the files on the volume. Such volumes include all disks, diskettes, and DECtapes.

Disk device An auxiliary storage device on which information can be read or written.

Display A peripheral device used to represent data graphically; normally refers to some type of cathode-ray tube system.

Downtime The time interval during which a device or system is inoperative.

Echo The printing of characters typed by the programmer on an I/O device such as a terminal.

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

Editor A program that allows the user to enter text into the computer and edit it. Editors are language-independent and will edit anything in character representation.

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 magnetic tape, which precedes the end of the reel.

Error Any discrepancy between a computed, observed, or measured quantity and the specified value or condition.

Execute To perform an instruction or run a program on the computer.

Extension The synonym used for file type.

External storage A storage medium other than main memory, for example, a disk or tape.

Field A specified area of a record used for a particular category of data.

FIFO First In/First Out, a data manipulation method in which the first item stored is the first item processed.

File A logical collection of data that is treated as a unit, occupies one or more blocks on a mass storage volume, and has an associated file name and type.

File maintenance The activity of keeping a mass storage volume and its directory up to date by adding, changing, or deleting files.

File name The alphanumeric character string assigned by a user to identify a file. It can be read by both an operating system and a user. A file name has a fixed maximum length that is system-dependent. (The maximum length in an RT–11 operation system is six characters, the first of which must be alphabetic. Spaces are not allowed.)

File specification A name that uniquely identifies a file maintained in any operating system. A file specification generally consists of at least three components: a device name, a file name, and a file type.

File-structured device A device on which data is organized into files. The device usually contains a directory of the files stored on the volume. (For example, a disk is a file-structured device, but a line printer is not.)

File type The alphanumeric character string assigned to a file either by an operating system or a user. 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.

Flowchart A graphical representation for the definition, analysis, or solution of a problem, in which symbols are used to represent operations, data, flow, and equipment.

Foreground The area in memory designated for use by a high-priority program. The program that gains the use of machine facilities immediately upon request.

FORTRAN IV FORmula TRANslation, a problem-oriented language designed to permit scientists and engineers to express mathematical operations in a form with which they are familiar. It is also used in a variety of applications, including process control, information retrieval, and commercial data processing.

Function An algorithm, accessible by name and contained in the system software, that performs commonly used operations.

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 as externals in the other modules that use them.

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 translated into more than one machine language instruction. Examples are BASIC–11 and FORTRAN–IV.

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

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 needs to work with. A symbolic instruction looks like ordinary language. Symbolic instructions must be changed into machine instructions 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.

Internal storage The storage facilities that form an integral physical part of the computer and that are directly controlled by the computer; for example, the registers of the machine and main memory.

Interpreter A computer program that translates and 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 and breaks the normal flow of the routine being executed.

Job A group of data and control statements that does a unit of work. A program and all of its related subroutines, data, and control statements is an example.

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

Library A file containing routines (macro definitions or relocatable object modules) 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.

Linkage The code that connects two separately coded routines and passes values and/or control between them.

Linked file A file whose blocks are joined together by references rather than by consecutive locations.

Linker A program that combines many relocatable object modules into an executable module. It satisfies global references and combines program sections.

Listing The printed copy generated by a line printer or terminal.

Load To store a program or data in memory. To place a volume on a device unit and put the unit on line.

Load map A table, produced by a linker, that provides information about a load module's characteristics; for example, the transfer address, the global symbol values, and the low and high limits of the relocatable code.

Load module A program in a format that is ready for loading and executing.

Location An address in storage or memory where a unit of data or an instruction can be stored.

Locked Pertaining to routines in memory that presently cannot be swapped or transferred.

Logical device name An alphanumeric name assigned by the user to represent a physical device. The name can then be used synonymously with the physical device name in all references to the device. Logical device names are used in device-independent systems to enable a program to refer to a logical device name assigned to a physical device at run-time.

Loop A sequence of instructions that is executed repeatedly until a terminal condition prevails.

Machine language The language used by the computer when performing operations.

Macro An instruction in a source language that is equivalent to a specified sequence of assembler instructions, or a command language that is equivalent to a specified sequence of commands.

Main program The module of a program that contains the instructions at which program execution begins. The main program usually exercises primary control over the operations performed; it also calls subroutines or subprograms to perform specific functions.

Mass storage Pertaining to a device that can store large amounts of data that are readily accessible to the computer.

Memory Any form of data storage, including main memory and mass storage, in which data can be read and written. Memory usually refers to main memory.

Memory image A replication of the contents of a portion of memory, usually in a file.

Mnemonic An alphabetic easy-to-remember representation of a function or machine instruction.

Monitor The master control program that observes, supervises, controls, or verifies the operation of a computer system. The collection of routines that controls the operation of user and system programs, schedules operations, allocates resources, performs I/O, and so forth.

Monitor command An instruction or command issued directly to a monitor from a user.

Monitor command mode The state of the operating system—indicated by a period at the left margin—that allows monitor commands to be entered from the terminal.

Mount a volume To logically associate a physical mass storage medium with a physical device unit. To place a volume on a physical device unit, for example, to place a magnetic tape on a magnetic tape drive and put the drive on the 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 magnetic tape and cassette.

Nonfile-structured device A device, such as a line printer or terminal, in which data cannot be organized as multiple files.

Object code Relocatable machine language code.

Object module The primary output of an assembler or compiler, which can be linked with other object modules and loaded into memory as an executable 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.

Octal Pertaining to the number system with a radix of eight; for example, octal 100 is decimal 64.

ODT On-line Debugging Technique, an interactive program for finding and correcting errors in programs.

Off-line Pertaining to equipment or devices not currently under direct control of the computer.

On-line Pertaining to equipment or devices directly connected to and under control of the computer.

Operand The data that an instruction operates upon. An operand is usually identified by an address part of an instruction.

Operating system The collection of programs, including a monitor and system programs, that organizes a central processor and peripheral devices into a working unit for the development and execution of application programs.

Operation The act specified by a single computer instruction. A program step undertaken or executed by a computer; for example, addition, multiplication, comparison. The operation is usually specified by the operator part of an instruction.

Operation code The part of a machine language instruction that identifies the operation the CPU is to perform.

Operator's console The set of switches and display lights used by an operator or a programmer to determine the status of the computer system and to start the computer.

Option An element of a command or command string that enables the user to select alternatives associated with the command. In the RT–11 operating system, an option consists of a slash character (/) followed by the option name and, optionally, a colon and an option value.

Output The result of a process; the transferring of data from internal storage to external storage.

Overflow A condition that occurs when a mathematical operation yields a result whose magnitude is larger than the hardware is capable of handling.

Overlay segment A section of code treated as a unit that can overlay code already in memory and be overlaid by other overlay segments when called from the root segment or another resident overlay segment.

Overlay structure A program overlay system consisting of a root segment and optionally one or more overlay segments.

Page The portion of a text file delimited by form feed characters and generally 50 to 60 lines long.

Parameter A variable that is given a constant value for a specific purpose or process.

Parity A binary digit appended to an array of binary digits to make the sum of all bits always odd or always even. It is used to check the validity of data.

PDP Programmable Data Processor.

Peripheral device Any device distinct from the computer that can provide input and/or accept output from the computer.

Physical device An I/O or peripheral storage device connected to or associated with a computer.

Priority A number, associated with a task, that determines the order in which the monitor will process the request for service by that task, relative to other tasks requesting service.

Process A set of related procedures and data that are executed and manipulated by a computer.

Processor In hardware, a data processor. In software, a computer program that includes the compiler, assembler, translator, and related functions for a specific programming language (for example, FORTRAN IV processor).

Program A set of machine instructions or symbolic statements combined to perform some task.

Programmed request A set of instructions (available only to programs) that is used to invoke a monitor service.

Program section A named, contiguous unit of code (instructions or data) that is considered as an entity and that can be relocated separately without destroying the logic of the program.

Radix The base of a number system; the number of digit symbols required by a number system.

RAM Random-Access Memory, memory that is accessed in such a way that the next location from which data is to be obtained is not dependent on the location of the data previously obtained.

ROM Read-Only Memory, memory whose contents are not alterable by computer instructions.

Real-time processing The computation performed while a related or controlled physical activity is occurring. The results of the computation can be used for guiding the process.

Record A collection of related items of data treated as a unit; for example, a line of source code.

Relative address The number that specifies the difference between the actual address and a base address.

Resident Pertaining to data or instructions that are permanently located in main memory.

Resource The computational power, programs, data files, storage capacity, or a combination of these that are available to a user.

Restart To resume execution of a program.

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.

Software The collection of programs and routines associated with a computer. Application programs, compilers, and library routines are examples.

Software bootstrap A bootstrap that is activated by loading the instructions of the bootstrap and specifying the appropriate load and start address.

Source code Text, usually in the form of an ASCII file, that represents a program. Such a file can be processed by the appropriate system program.

Source language The system of symbols and syntax used to describe a procedure that a computer can execute.

Storage Pertaining to a device into which data can be entered, in which it can be held, and from which it can be retrieved at a later time.

String A connected sequence of entities, such as a line of characters.

Subprogram A program or a sequence of instructions that can be called to perform the same task (though perhaps on different data) at different points in a program, or in different programs.

Subroutine See Subprogram.

Swapping The process of moving data from memory to a mass storage device, temporarily using the empty memory area for another purpose, and then restoring the original data to memory.

Synchronous Pertaining to related events where all changes occur simultaneously or in definite timed intervals.

Syntax The structure of expressions in a language and the rules governing the structure of a language.

System program A program that performs system-level functions. A program that is part of the basic operating system (for example, a system utility program) is a system program.

System volume The volume on which the operating system is stored.

Table A collection of data in a well-defined list.

Terminal An I/O device, such as a VT100 terminal, that includes a keyboard and a display mechanism. In PDP–11 systems, a terminal is used as the primary communication device between a computer system and a user.

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.

Word Sixteen binary digits treated as a unit in PDP–11 computer memory.

Write-enabled The condition of a volume that allows information to be written on it.

Write-protected The condition of a volume that protects the volume against information being written on it.

Index

$19.00

The first of four books in the RT-11 Technical User's Series, *Working with RT-11* introduces programmers to the components and features of Digital's single-user, real-time operating system, RT-11. The book describes the keyboard functions on both VT100 and Professional 300 terminals. It examines the commands and utilities needed to edit, organize, and maintain files and develop programs from start to finish. Practice exercises encourage programmers using RT-11 for the first time to sit down at their terminals and become acquainted with the system.

The authors develop courses for Educational Services Division of Digital Equipment Corporation in Reading, England.

For a complete list of DECbooks, write to:

Digital Press
Digital Equipment Corporation
12 Crosby Drive, Bedford, MA 01730

ISBN 0-932376-31-2