pdp11

RT-11
System User's Guide

Order No. DEC-11-ORGDA-A-D

digital

CONTENTS

CONTENTS (Cont.)

CONTENTS (Cont.)

CONTENTS (Cont.)

CONTENTS (Cont.)

CONTENTS (Cont.)

CONTENTS (Cont.)

CONTENTS (Cont.)

CONTENTS (Cont.)

FIGURES

CONTENTS (Cont.)

FIGURES (Cont.)

TABLES

TABLES (Cont.)

PREFACE

This manual describes how to use the RT-11 system; it provides enough information for you to perform ordinary tasks such as program development, program execution, and file maintenance. This manual is appropriate for you if you are already familiar with computer software fundamentals and have some experience using RT-11. If you have no RT-11 experience, you should read the *Introduction to RT-11* before consulting this manual. If you have experience with an earlier release of RT-11 (this is version 3), you should read the *RT-11 System Release Notes* to learn how RT-11 V03 differs from earlier versions. If you are interested in more sophisticated programming techniques or in system programming, you should read this manual first and then proceed to the *RT-11 Advanced Programmer's Guide*.

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

CHAPTER SUMMARY

The first two chapters make up Part I of this manual, RT-11 Overview. Read Part I to gain an understanding of the RT-11 system as a whole.

Chapter 1 describes the program development process in general as well as the system software and hardware components.

Chapter 2 describes the three monitors that are available with an RT-11 system.

Chapters 3 and 4 compose Part II of the manual, System Communication. Read Part II to become familiar with RT-11 system conventions and to learn how to interact with the RT-11 monitor directly from the console terminal.

Chapter 3 describes system conventions, such as data formats, file naming conventions, and terminal keyboard special functions.

Chapter 4 introduces the keyboard monitor commands. These important commands are your means of communicating with the monitor and performing computer tasks.

Part III, Text Editing, consists of Chapter 5, EDIT. Read Chapter 5 to learn how to manipulate text on the RT-11 system.

Part IV, Utility Programs, consists of 10 chapters that describe the many programs provided with the RT-11 system. If you are an advanced user, you may want to read Chapters 6 through 15 to learn about the RT-11 system programs in detail. However, if you are a new user or primarily a high-level language programmer, you do not have to understand how these system programs work to make use of them through the monitor command language (described in Chapter 4).

Chapter 6 describes the Command String Interpreter and explains the command syntax you use to communicate with the RT-11 system programs.

Chapters 7 through 9 describe the RT-11 system utility programs, PIP, DUP, and DIR.

Chapter 10 describes MACRO, the RT-11 assembly language.

Chapters 11 through 15 describe the RT-11 system utility programs, LINK, LIBR, DUMP, FILEX, and SRCCOM.

Part V, Altering Assembled Programs, explains the use of some sophisticated programming tools.

Chapters 16 through 18 describe the RT-11 programs, ODT, PATCH, and PAT. These three programs can help you debug programs and make changes to programs that are already assembled.

Appendix A contains a description of RT-11 BATCH. Appendix B contains a table of the keyboard monitor commands, their abbreviations, and their system program equivalents.

DOCUMENTATION CONVENTIONS

This section describes the symbolic conventions used throughout this manual. Familiarize yourself with these conventions before you continue reading the manual.

Conventions used in this manual include the following items:

1. Examples consist of actual computer output wherever possible. In the examples, responses entered by a user are shown in red to distinguish them from computer output, which is shown in black.
2. Unless the manual indicates otherwise, terminate all commands or command strings with a carriage return. Where necessary, this manual uses the symbol (RET) to represent a carriage return, (LF) to represent a line feed, (SP) for a space, and (TAB) to represent a tab.
3. Terminal and console terminal are general terms used throughout all RT-11 documentation to represent any terminal device, including DECwriters, displays, and Teletypes[1].
4. To produce several characters in system commands you must type a combination of keys concurrently. For example, hold down the CTRL key and type O at the same time to produce the CTRL/O character. Key combinations such as this one are documented as CTRL/O, CTRL/C, SHIFT/N, etc.
5. In descriptions of command syntax, capital letters represent the command name, which you must type. Lower case letters represent a variable, for which you must supply a value.

   Square brackets [ ] enclose optional choices; you can include the item in brackets, or you can omit it, as you choose.

   Braces { } enclose a group of options from which you can choose only one.

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

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

   The following is a typical example of command syntax:

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

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

   ```
   .DELETE/NOQUERY DT1:MYFILE.FOR
   ```

[1] Teletype is a registered trademark of the Teletype Corporation.

PART I

RT-11 OVERVIEW

RT-11 is a single-user programming and operating system for the PDP-11 series of computers. This system can use a wide range of peripherals and can access up to 124K (126,976) words of either solid state or core memory. (4K words of the maximum 128K (131,072) words of memory are reserved for device interfacing.)

Three system monitors are provided by RT-11: the single-job monitor (SJ), the foreground/background monitor (FB), and the extended memory monitor (XM).

The single-job monitor allows one program at a time to reside in memory. The program executes until it completes or until you interrupt it with a keyboard command.

The foreground/background monitor allows two independent programs to reside in memory at one time. The foreground program, however, takes priority over the background program. RT-11 allows the background program to execute whenever the foreground program is in a wait state. Typically, the foreground program performs a time-dependent task, such as sampling material every few seconds and then analyzing the resultant data. A background program, on the other hand, usually performs a time-independent task, such as file maintenance or program development. This sharing of resources between two tasks greatly increases the efficiency of your RT-11 system.

The extended memory monitor provides all the features of the foreground/background monitor and, in addition, allows you to access up to 124K (126,976) words of memory. The other two monitors are restricted to 28K words of main memory. (4K words of the 32K words of memory available are reserved for device interfacing.)

These three monitors are upward compatible. That is, the foreground/background monitor provides all the features of the single-job monitor, and the extended memory monitor offers all the features of the foreground/background monitor.

You control the RT-11 system from the console terminal. The monitor commands that you use to direct the system are described in Chapter 4 of this manual.

In addition to the three monitors, RT-11 provides a full complement of system programs that can perform some more specific tasks than the keyboard monitor commands can. If you are an average user, though, the keyboard monitor commands should be sufficient for your needs. There is a summary of the system programs in Section 1.2; they are described in more detail in individual chapters of this manual.

RT-11 also supports a variety of language processors including MACRO-11, an assembly language, and several high-level languages such as FORTRAN IV and BASIC.

The following two chapters describe system software and hardware components, program development, and the three RT-11 monitors.

CHAPTER 1
SYSTEM COMPONENTS

This chapter describes briefly the software and hardware components available for you to use with the RT-11 system. The software components include the text editor and the many system programs that perform specific tasks. The hardware components include system clocks, printing and display terminals, external storage devices (such as magnetic tape drives), and other peripheral devices (such as card readers and line printers).

1.1 PROGRAM DEVELOPMENT

Computer systems (such as RT-11) are ideal for program development. You can make use of the programming tools available on your system to develop programs to suit your needs. The number and type of tools available on any given system depend on many factors (including the size of the system, its application, and its cost). Most DIGITAL systems, however, provide several basic program development aids. These aids generally include an editor, an assembler, a linker, a debugger, and a librarian. A high level language, such as FORTRAN or BASIC, is also usually available.

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

If you use the editor to write a source program, development does not stop with the creation of this program. Since the computer cannot understand any language but machine language (which is a set of binary command codes), you need an intermediary program to convert source code into the instructions the computer can execute. This is the function of an assembler or language translator.

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

The assembler produces output called object output because it is composed of object (or binary) code. On PDP-11 systems, the object output is called a module; it contains your source program in the binary language that is acceptable to a PDP-11 computer.

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

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

You can very rarely create a program that does not contain at least one unintentional error, either in the logic of the program or in its coding. You may discover errors while you are editing your program, or the assembler may find errors

during the assembly process and inform you by means of error codes. The linker may also catch certain errors and issue appropriate messages. Often, however, it is not until execution that you discover that your program is not working properly. Programming errors may be extremely difficult to find, and for this reason, a debugging tool is usually available to aid you in determining the cause of your error.

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

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

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

These are a few of the programming tools offered by most computer systems. The next section summarizes specific programming aids available to you as an RT-11 user.

1.2 SYSTEM SOFTWARE COMPONENTS

The following is a brief summary of the specific system programs and programming available to you as an RT-11 user:

1. The keyboard monitor commands (described in Chapter 4) are your means of controlling the system. You can use these English-language commands to perform file maintenance, library maintenance, handler modification, program development, and program execution. If you are an average user, the keyboard monitor commands should be sufficient for your needs.
2. The text editor (EDIT, described in Chapter 5) creates or modifies source files for use as input to language-processing programs such as the assembler or FORTRAN. EDIT contains text manipulation commands that permit quick and easy editing of a text file. EDIT also allows you to use a VT11 or VS60 display processor if one is part of the hardware configuration.
3. The peripheral interchange program (PIP, described in Chapter 7) is the RT-11 file maintenance program. It transfers files among all devices that are part of the RT-11 system and renames or deletes files.
4. The device utility program (DUP, described in Chapter 8) performs general device utilities such as initializing devices, duplicating their contents, and reorganizing files on the devices. It operates only on RT-11 file-structured devices.
5. The directory program (DIR, described in Chapter 9) produces directory listings.
6. The MACRO assembler (described in Chapter 10) is a 2-pass assembler that assembles one or more ASCII source files of statements and assembler language instructions into a single binary object file.
7. The linker (LINK, described in Chapter 11) converts a collection of object modules from compiled or assembled programs and subroutines into a memory image file that RT-11 can load and execute. LINK provides some optional features that:

    a. Search library files for subroutines that you specify
    b. Produce a load map that lists the assigned absolute addresses
    c. Provide overlay capabilities to very large programs
    d. Produce files suitable for execution in the foreground.

8. The librarian (LIBR, described in Chapter 12) lets you create and maintain libraries of functions and routines. These routines are stored on a random access device in library files, where the linker can reference them. You can also create MACRO libraries to be used by the MACRO assembler.
9. DUMP (described in Chapter 13) prints for examination all or any part of a file in octal words, octal bytes, ASCII and/or Radix-50 characters.
10. The file exchange utility (FILEX, described in Chapter 14) transfers files between DECsystem-10, PDP-11 RSTS, and DOS BATCH on DECtape and disks, and between RT-11 and IBM systems on diskettes.
11. The source compare utility (SRCCOM, described in Chapter 15) performs a character-by-character comparison of two ASCII text files. You can request that the differences be listed in an output file or directly on the line printer or terminal to ensure that edits have been performed correctly.
12. On-line debugging technique (ODT, described in Chapter 16) aids you in debugging assembled and linked object programs. It can:

    a. Print and optionally change the contents of specified locations
    b. Execute all or part of the object program
    c. Single-step through the program
    d. Search the object program for bit patterns.

13. The patching utility program (PATCH, described in Chapter 17) performs minor modifications to memory image files (output files produced by the linker).
14. The object module patching program (PAT, described in Chapter 18) performs minor modifications to files in object format (output files produced by the FORTRAN compiler or the MACRO assembler). It can merge several object files into one.
15. The RT-11 FORTRAN system subroutine library (described in the *RT-11 Advanced Programmer's Guide*) is a collection of FORTRAN callable routines that make the programmed requests and various utility functions available to you as a FORTRAN programmer. This library also provides a string manipulation package and 2-word integer package for RT-11 FORTRAN.
16. BATCH (Appendix A) is a complete job-control language that allows RT-11 to operate unattended.

1.3 SYSTEM HARDWARE COMPONENTS

The smallest RT-11 system, one that uses the SJ monitor exclusively, requires a PDP-11 series computer with at least 8K words of memory, a random-access device, and a console terminal. The addition of the FB monitor requires another 8K words of memory and either a line frequency or a programmable clock. The addition of the XM monitor requires a KT11 memory management unit and still another 8K words of memory.

The RT-11 operating system adapts itself to take advantage of any amount of memory on a system and does not need to be reconfigured for a particular memory size. The SJ monitor operates in systems ranging from 8K words to 28K words in memory size. The FB monitor operates in systems ranging from 16K words to 28K words in memory size. The XM monitor operates in systems ranging from 24K words to 124K (126,976) words in memory size.

Table 1-1 lists the devices that RT-11 supports.

Table 1-1 RT-11 Hardware Components

Type	Controller	Device
Disk		
Cartridge	RK11 RK611	RK05/RK05F RK06
Fixed-head	RF11 RH11	RS11 RJS03, RJS04
Removable Pack	RP11	RP02, RP03
Diskette	RX11	RX01
DECtape	TC11	TU56
Magtape	TM11/TMA11 RH11	TU10, TS03 TJU16, TU45
Cassette	TA11	TU60
High-Speed Paper Tape Reader/Punch	PC11 PR11	PC11 (both) PR11 (reader only)
Line Printer	LS11 LV11 LP11	LS11, LA180 LV11 (printer only) all LP11 controlled printers
Card Reader	CR11 CM11	CR11 CM11
Terminal	DL11	LT33, LT35, LA30P, LA36, LA120, VT50, VT52, VT55, VT05 VT61
Display Processor	VT11 VS60	VR14-L, VR17-L
Clock		KW11-L, KW11-P
Terminal and Clock	DL11-W	terminal/clock combination

CHAPTER 2
OPERATING ENVIRONMENTS

The RT-11 system offers three complete operating environments: single-job (SJ) operation, foreground/background (FB) operation, and extended memory (XM) operation. You control each environment with the appropriate monitor: SJ, FB, and XM.

You must define your needs before deciding which environment to use and consequently which monitor to run. The following sections provide information to help you ascertain which monitor is suitable for your application.

2.1 RT-11 SINGLE-JOB MONITOR

The RT-11 single-job monitor provides a single-user, single-program system that can operate in as little as 8K words of memory. The SJ monitor is useful for extensive program development; since the monitor itself requires only 2K words of memory, there are at least 6K words left for your program and its buffers and tables. The SJ environment is also suitable for running programs that require a high data transfer rate, since the SJ monitor services interrupts quickly.

You can use all the system programs (listed in Section 1.2) under the SJ monitor. Monitor commands and programmed requests are also available to you as an SJ user.

In summary, the SJ monitor is smaller and faster than the FB and XM monitors; it is most useful when you are concerned with program size versus available memory and when you need a dedicated system.

2.2 RT-11 FOREGROUND/BACKGROUND MONITOR

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

In a foreground/background system, the foreground job is the time-critical, real-time job, and the FB monitor gives it priority over the background job. Whenever the foreground job reaches a state in which no useful processing can be done until some external event occurs, the monitor executes the background job, if possible. The background job then runs until the foreground job is again ready to execute. The processor then interrupts the background job and resumes the foreground job.

In effect, the RT-11 foreground/background monitor allows a time-dependent job to run in the foreground while a time-independent job, such as program development, runs in the background. All RT-11 system programs can run as the background job in a FB system. Thus, you can run FORTRAN, BASIC, MACRO, etc. in the background while the foreground is collecting, storing, and analyzing data. In addition, the FB monitor gives you the ability to set timer routines, suspend and resume FB jobs, and send data and messages between the two jobs. The FB monitor is most often used for laboratory work, data acquisition, and real-time applications.

You can link most of the programs you write for an RT-11 system to run as foreground jobs. There are a few coding restrictions, which are explained in the *RT-11 Advanced Programmer's Guide*. A foreground program has access to all of the features available to the background job (opening and closing files, reading and writing data, etc.).

2.3 RT-11 EXTENDED MEMORY MONITOR

The extended memory monitor (XM) is an extension of the foreground/background (FB) environment. Generally, references in this manual to FB operation also apply to XM operation. The single-job monitor does not support

extended memory. The XM monitor permits either foreground or background jobs to extend their effective logical program space beyond the 32K word restriction imposed by the 16-bit address word of the PDP-11 processors. The XM monitor manages extended memory space as a system resource and dynamically allocates it as you request. A program can map selected portions of its addressing space into extended memory by means of a set of programmed requests. A detailed description of extended memory and how to use it appears in the *RT-11 Advanced Programmer's Guide*.

2.4 FACILITIES AVAILABLE ONLY IN RT-11 FB

Some features available to you as a FB user include:

1. **Mark time.** The .MRKT programmed request allows your program to set clock timers for specified amounts of time. When the timer runs out, the system enters the routine that you specify. You can enter as many mark time requests as you need, providing that you reserve system queue space. The mark time feature is available to SJ monitor users as a SYSGEN option.
2. **Timed wait.** The .TWAIT programmed request allows your program to "sleep" until a period of time that you specify elapses. A foreground program, for example, may need to act on sample data and write it to mass storage once every few minutes. While the foreground program is idle, the background program can run.
3. **Send data, receive data.** The .SDAT and .RCVD programmed requests permit the foreground and background programs to communicate with each other. The send and receive data functions let one program send messages or data of variable size blocks to the other program. For example, you can transfer data directly from a foreground collection program to a background analysis program.
4. **Channel copy.** The .CHCOPY programmed request allows two programs to share the same data file.
5. **Device.** The .DEVICE programmed request allows you to turn off specific devices upon program termination.
6. **Protect.** The .PROTECT programmed request lets you protect the vectors that one program uses from interference by another program.
7. **Channel status.** The .CSTAT programmed request returns status data about an open channel.

You can learn more about these programmed requests and how to use them in Chapter 2 of the *RT-11 Advanced Programmer's Guide*.

2.5 FACILITIES AVAILABLE ONLY IN RT-11 XM

An optional extension of the FB environment is the extended memory monitor (XM), which permits you to extend the logical address space for either foreground or background jobs. Some features available to you only when you use the XM monitor are:

1. **Create a region.** The .CRRG programmed request allows you to allocate a region in extended memory for the current program.
2. **Eliminate a region.** The .ELRG programmed request eliminates an extended memory region and returns it to the free list so it can be used by other programs.
3. **Create an address window.** The .CRAW programmed request unmaps and eliminates conflicting address windows, creates new windows to address extended memory, and maps new windows to the regions you specify. It directs the monitor to give the program a window into the region it has created. This request allows the program to access the physical memory as if it were local to the program.
4. **Eliminate an address window.** The .ELAW programmed request unmaps and eliminates address windows.
5. **Map.** The .MAP programmed request lets you map and remap windows.
6. **Status.** The .GMCX programmed request returns status data about window mapping.
7. **Unmap.** The .UNMAP programmed request lets you unmap a window.

You can learn more about these programmed requests and how to use them in Chapter 3 of the *RT-11 Advanced Programmer's Guide*.

PART II
SYSTEM COMMUNICATION

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

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

Programmed requests (described in detail in Chapter 2 of the *RT-11 Advanced Programmer's Guide*) are source program instructions that request the monitor to perform monitor services. These instructions allow assembly language programs to use the available monitor features. A running program communicates with the monitor through programmed requests. FORTRAN programs have access to programmed requests through the system subroutine library. Programmed requests can, for example, manipulate files, perform input and output, and suspend and resume program operations.

The two chapters in this part describe system conventions and contain information that helps you get started with RT-11. Chapter 4 introduces the keyboard monitor commands, which are your means of controlling the RT-11 system.

CHAPTER 3
SYSTEM CONVENTIONS

This chapter contains information to help you start using the RT-11 system. It describes:

- Startup procedure
- Data formats
- Physical device names
- File names and file types
- Device structures
- Special function keys
- Foreground input and output
- Monitor type-ahead feature

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

3.1 SYSTEM STARTUP

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

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

```
RT-11SJ Vnnx-nnx
RT-11FB Vnnx-nnx
RT-11XM Vnnx-nnx
```

The message that prints indicates which monitor (SJ, FB, or XM) is loaded; you establish which is to be loaded during the system build operation.

Vnnx represents the version and release number of the monitor -- for example, V03, for Version 3 (release A). nnx represents the library submission number and the patch level -- for example, 01B, for library number 1 (patch level B).

As soon as a monitor takes control of the system, it attempts to execute keyboard monitor commands from an indirect file called STARTS.COM for the SJ monitor, STARTF.COM for the FB monitor, and STARTX.COM for the XM monitor. You can place commands in this startup file to perform routine tasks for you, such as assigning logical device names to physical devices or setting the current date. (Indirect files are discussed in Section 4.3.) If the monitor does not find the appropriate file, it issues a warning message. The system then prints its prompt (.) indicating that it is ready to accept commands. You should now write-enable the system device.

To bring up an alternate monitor while under control of the one currently running, use the BOOT command described in Section 4.4 of this manual.

3.2 DATA FORMATS

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

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

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

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

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

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

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

There are a number of other types of binary data that different parts of the RT-11 system use in addition to the more common types listed here.

3.3 PHYSICAL DEVICE NAMES

When you request services from the monitor, it is sometimes necessary to specify a physical peripheral device on which the service is to be performed. You can reference devices by means of a standard 2-character device name. Table 3-1 lists each name and its related device. If you do not specify a unit number for devices with more than one unit, the system assumes unit 0.

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

3.4 FILE NAMES AND FILE TYPES

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

3.5 DEVICE STRUCTURES

RT-11 devices are categorized according to two characteristics: 1) the device's physical structure and 2) the device's method of processing information. All RT-11 devices are either randomly accessed or sequentially accessed.

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

Table 3-1 Permanent Device Names

Permanent Name	I/O Device
CR:	CR11/CM11 Card Reader
CTn:	TA11 Cassette (n is 0 or 1)
DK:	The default logical storage device for all files. DK: is initially the same as SY:
DKn:	The specified unit of the same device type as DK: if DK: is unassigned
DMn:	RK06 Disk (n is an integer in the range 0-7)
DPn:	RP02, RP03 Disk (n is an integer in the range 0-7)
DSn:	RJS03/4 Fixed-Head Disks (n is an integer in the range 0-7)
DTn:	DECtape (n is an integer in the range 0-7)
DXn:	RX01 Floppy Disk (n is an integer in the range 0-3)
EL:	Error Logging Handler
LP:	Line Printer
MMn:	TJU16/TU45 (industry compatible) Magtape (n is an integer in the range 0-7)
MTn:	TM11/TMA11/TS03 (industry compatible) Magtape (n is an integer in the range 0-7)
NL:	Null device
PC:	PC11 combined High-Speed Paper Tape Reader and Punch
RF:	RF11 Fixed-Head Disk Drive
RKn:	RK05 Disk Cartridge Drive (n is an integer in the range 0-7)
SY:	The default logical system device; the device and unit from which the system is bootstrapped
SYn:	The specified unit of the same device type as SY: if SY: is unassigned
TT:	Console Terminal Keyboard and Printer

Table 3-2 Standard File Types

File Type	Meaning
.BAD	Files with bad (unreadable) blocks; you can assign this file type whenever bad areas occur on a device. The .BAD file type makes the file permanent in that area, preventing other files from using it and consequently becoming unreadable
.BAK	Editor backup file
.BAS	BASIC source file (BASIC input)
.BAT	BATCH command file
.COM	Indirect file
.CTL	BATCH control file generated by the BATCH compiler
.CTT	BATCH internal temporary file
.DAT	BASIC or FORTRAN data file
.DBL	DIBOL source file
.DIF	SRCCOM output file
.DIR	Directory listing file
.DMP	DUMP output file
.FOR	FORTRAN IV source file (FORTRAN input)
.LDA	Absolute binary file (optional linker output)
.LOG	BATCH log file
.LST	Listing file (MACRO, FORTRAN, LIBR, or DIBOL output)
.MAC	MACRO source file (MACRO or SRCCOM input)
.MAP	Map file (linker output)
.OBJ	Relocatable binary file (MACRO or FORTRAN output, linker input, LIBR input and output)
.REL	Foreground job relocatable image (linker output, default for monitor FRUN command)
.SAV	Memory image; default for R, RUN, SAVE and GET keyboard monitor commands; also default for linker output
.SML	System MACRO library
.SOU	Temporary source file generated by BATCH
.STB	Symbol table file in object format containing all the symbols produced during a link
.SYS	System files and handlers

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

File-structured devices are those devices that allow the system to store data under assigned file names. RT-11 devices that are file-structured include all disk, DECtape, magtape, and cassette devices. Non-file-structured devices, however, contain a single logical collection of data. These devices, including the line printer, card reader, terminal, and paper tape reader and punch, are generally used for reading and listing information.

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

The *RT-11 Software Support Manual* explains methods of interfacing a device with a user-defined directory structure to the RT-11 system.

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

Table 3-3 Device Structures

Device	Random-Access	Sequential-Access	File-Structured	Non-file-Structured	RT-11 directory-Structured	Non-RT-11 directory-Structured
Disk	x		x		x	
DECtape	x		x		x	
Magtape		x	x			x
Cassette		x	x			x
Paper tape		x		x		
Card reader		x		x		
Line printer		x		x		
Terminal		x		x		

3.6 SPECIAL FUNCTION KEYS

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

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

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

3.7 FOREGROUND/BACKGROUND TERMINAL I/O

Console input and output under FB are independent functions; therefore, you can type input to one job while another job prints output. You may be in the process of typing input to one job when the system is ready to print output from the other job on the terminal. In this case, the job that is ready to print interrupts you and prints the message on the terminal; the system does not redirect input control to this job, however, unless you type a CTRL/B or CTRL/F. If

Table 3-4 Special Function Keys

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

(Continued on next page)

Table 3-4 (Cont.) Special Function Keys

Key	Function
CTRL/Q	CTRL/Q resumes printing characters on the terminal from the point printing previously stopped because of a CTRL/S. CTRL/Q does not echo and has no special meaning under the FB monitor if a SET TT NOPAGE command is in effect.
CTRL/S	CTRL/S temporarily suspends output to the terminal until you type a CTRL/Q. CTRL/S does not echo. Under the FB monitor, CTRL/S is not intercepted by the monitor if TT NOPAGE is in effect.
CTRL/U	CTRL/U deletes the current input line and echoes as ^U followed by a carriage return at the terminal. (The current line is defined as all characters back to, but not including, the most recent line feed, CTRL/C, or CTRL/Z.)
CTRL/Z	CTRL/Z terminates input when used with the terminal device handler (TT). It echoes ^Z on the terminal. The CTRL/Z itself does not appear in the input buffer. If TT is not being used, CTRL/Z has no special meaning.
DELETE or RUBOUT	DELETE deletes the last character from the current line and echoes a backslash plus the character deleted. Each succeeding DELETE deletes and echoes another character. The system prints an enclosing backslash when you type a key other than DELETE. This erasure is performed from right to left up to the beginning of the current line. If you are using a video display terminal, DELETE deletes characters with a backspace, space, backspace sequence. Your corrections appear on the screen; RUBOUT does not enclose them with backslash characters.

you type input to one job while the other has output control, the system suppresses the echo of the input until the job accepting input gains output control; at this point, all accumulated input echoes.

If the foreground job and background job are ready to print output at the same time, the foreground job has priority. The system prints output from the foreground job until it encounters a line feed. At that point, output from the background job prints until a line feed is encountered, and so forth.

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

3.8 TYPE-AHEAD FEATURE

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

```
.DIRECTORY/PRINTER
DATE
```

While the first command line is executing, you can type the second line. The system stores this terminal input in a buffer and uses it when the system completes the first operation.

If you type a single CTRL/C while the system is in this mode, the system puts CTRL/C into the buffer. The program currently executing exits when you make a terminal input request. Typing a double CTRL/C returns control to the monitor immediately.

If type-ahead input exceeds the input butter capacity (usually 80 characters), the terminal bell rings and the system accepts no characters until a program uses part of the type-ahead buffer, or until you delete characters. No input is lost. Type-ahead is particularly useful when you specify multiple command lines to system programs. If you terminate a job by typing two CTRL/Cs, the system discards any unprocessed type-ahead.

If you use type-ahead with EDIT or BASIC, the system does not echo characters on the terminal but stores them in the buffer until the system processes a new command. The program echoes the characters only when it actually uses them.

CHAPTER 4
INTERACTIVE COMMANDS

Keyboard commands allow you to communicate with the RT-11 system. You enter keyboard commands at the terminal and the operating system immediately acknowledges and acts upon these requests.

4.1 COMMAND SYNTAX

This section describes the syntax conventions this manual uses to discuss the monitor command language. The Preface to this manual contains a more detailed list of the symbolic conventions used throughout the manual. You should familiarize yourself with the symbols and their meanings before you continue reading this chapter.

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

The general syntax for a command is:

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

or

```
COMMAND[/option. . .]
PROMPT1? input-filespec[/option. . .]
PROMPT2? output-filespec[/option. . .]
```

where

COMMAND	is the command name.
/option	represents a command qualifier that specifies the exact action to be taken. Any option you supply here applies to the entire command string.
input-filespec	represents the file on which the action is to be taken.
/option	represents a file qualifier that specifies more detailed information about that particular file.
output-filespec	represents the file that is to receive the results of the operation.
/option	represents a file qualifier that specifies more detailed information about that particular file.

This manual provides a graphic illustration to clarify the syntax for each of the keyboard monitor commands. See Figure 4-1 for an illustration of a typical command. The illustrations provide a ready-reference list of the options that the commands accept, as well as information that makes the commands easier to use. The following list describes the conventions that are used in the illustrations.

1. Capital letters represent command names or options, which you must type as shown. (Abbreviations are discussed later in Section 4.1.)
2. Lower case letters represent arguments or variables for which you must supply values. For options that accept numeric arguments, the system interprets the values as decimal, unless otherwise stated. Some values, usually memory addresses, are interpreted as octal; these cases are noted in the accompanying text.
3. Square brackets [] enclose optional choices; you can include the item that is enclosed in the brackets or you can omit it, as you choose. If a vertical list of items is enclosed in square brackets, you can combine the options that appear in the list. However, if an option is set off from the others by blank lines (see /BOOT and /DEVICE in Figure 4-1), you cannot combine that option with any other option in the list.
4. Braces { } enclose options that are mutually exclusive. You can choose only one option from a group of options that appear in braces.
5. It is conventional to place command options (those qualifiers that apply to the entire command line) immediately after the command. However, it is also acceptable to specify a command option after a file specification. File options (those that qualify a particular file specification) must appear in the command line directly after the file to which they apply. The illustration for each command shows which options are file qualifiers, and whether they must follow input or output file specifications.
6. A line such as [NO]QUERY represents two mutually exclusive options: QUERY and NOQUERY.
7. Underlining indicates default options.

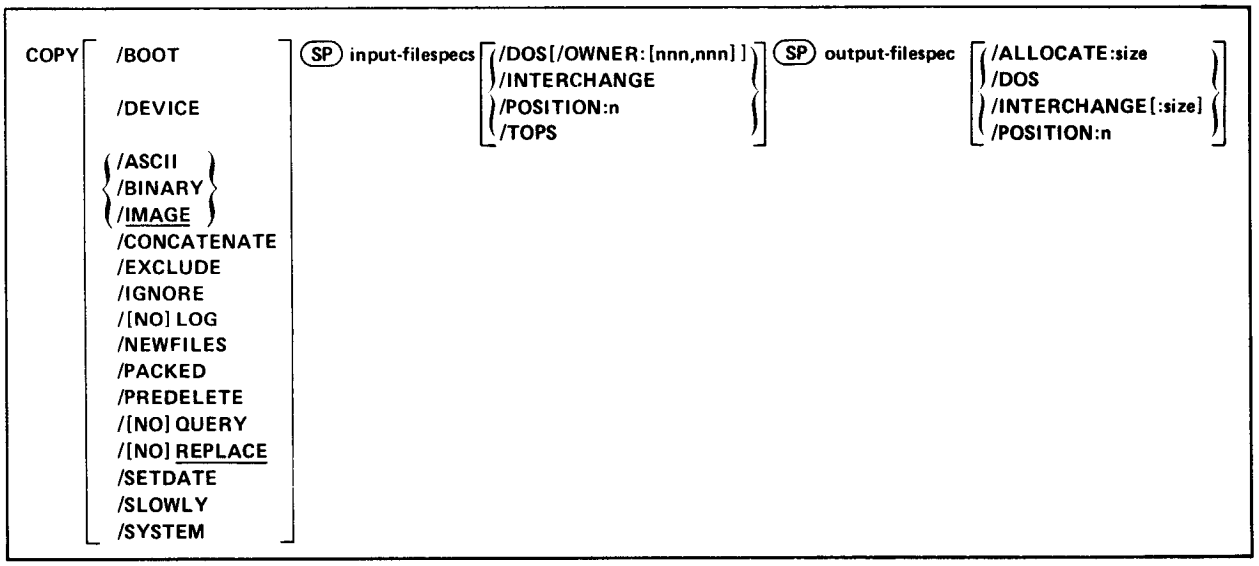


Figure 4-1 Sample Command Syntax Illustration

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

dev:filnam.typ

where

dev:	represents either a logical device name or a physical device name, which is a two- or three-character name from Table 3-1.
filnam	represents the one- to six-character alphanumeric name of the file.
.typ	represents the one- to three-character alphanumeric file type, some of which are listed in Table 3-2.

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

```
DX1:TEST.LST
```

You can omit the device name:

```
TEST.LST
```

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

If you want to specify several files on the same device, you can use a technique called factoring:

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

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

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

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

NOTE

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

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

```
RUN TEST.
```

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

You can enter up to six input files and up to three output files for some commands. If the command string does not fit on one line of your terminal, use the hyphen (-), followed by a carriage return, as a continuation character and break the string into smaller sections. Use a carriage return to terminate the command string.

Some of the command and file qualifiers are mutually exclusive options. You should avoid using a combination of options that gives contradictory instructions to the system. For example:

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

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

The keyboard monitor commands are all English-language words. This feature makes the commands easier for you to understand and use. However, it can become tedious to type words like CROSSREFERENCE and ALLOCATE frequently. You can use as abbreviations the minimum number of characters that are needed to make the command or option unique. Table B-1 in Appendix B lists the minimum abbreviations for the commands and options.

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

CONCATENATE can be shortened to CONC
NOCONCATENATE can be shortened to NOCONC

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

```
C TEST.LST
```

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

```
COPY/CONCATENATE
From?
```

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

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

The system prompts you for an output file specification:

```
To ?
```

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

```
DX2:TEST.LST
```

The command now executes.

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

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

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

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

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

4.2 WILDCARDS

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

An asterisk (*) can replace a file name:

```
*.MAC
```

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

An asterisk (*) can replace a file type:

```
TEST.*
```

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

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

```
*.*
```

The system selects all files on device DK:.

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

```
A*B.MAC
```

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

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

```
A%B.MAC
```

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

Table 4-1 lists commands that support wildcards.

Table 4-1 Commands Supporting Wildcards

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

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

Table 4-2 Wildcard Defaults

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

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

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

The system interprets this command to mean:

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

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

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

```
.DIRECTORY
```

The system interprets this command to mean:

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

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

```
.PRINT  DK:
```

The system interprets this command to mean:

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

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

```
.DELETE/NOQUERY  MYPROG
```

The system interprets this to mean:

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

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

4.3 INDIRECT FILES

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

4.3.1 Creating Indirect Files

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

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

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

In addition to using the keyboard monitor commands, you can also run one of the RT-11 system utility programs in an indirect file. In this case, structure your input to conform to the Command String Interpreter syntax described in Chapter 6. The following file starts the directory system utility program and lists the directory of two devices on the line printer.

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

Note that the last command line is ^C. This is not the standard CTRL/C sequence you enter by holding down the CTRL key and typing a C. Rather, it is a readable CTRL/C that consists of two separate characters: a circumflex (uparrow)

followed by a C. This sequence represents CTRL/C in indirect files because the two-character sequence is easier to read if you list the contents of the indirect file with the PRINT or TYPE command. This two-character sequence terminates the directory program so that control returns to the monitor when the indirect file finishes executing. Otherwise, the directory program would be left waiting for input from the console terminal when the indirect file finishes executing.

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

Some commands normally require a response from you as they execute. The INITIALIZE command, for example, prints the ARE YOU SURE? message and waits for you to type Y and a carriage return before it executes. The DELETE command requests confirmation from you before it deletes a file. There are three ways to control interaction with the executing command. One way is to use the /NOQUERY option on each command that allows it. This option suppresses the confirmation messages entirely when you use the command in an indirect file. A second procedure is suitable for a command like INITIALIZE, which has only one confirmation query. INITIALIZE can accept your response from within the indirect file. Place the Y response on a separate line in the indirect file, as the following example shows.

```
INITIALIZE/DOS DT1:
Y
```

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

There is yet another way to deal with commands that require a response from you. Both the INITIALIZE and LINK commands have options that prompt you for data. This section describes two methods of responding to these prompts, when more than just a Y response is required.

The INITIALIZE command with the /VOLUMEID option permits you to specify a volume ID and owner name for a device. You can place your responses in the indirect file, as this example shows:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```
!INDIRECT FILE TO ASSEMBLE THE MONITOR
R MACRO
RK1:RT11SJ=SJ,SYCND,KMON,USR,RMONSJ,KMOVLY
RK1:RKBTSJ=SJ,SYCND,BSTRAP    !ASSEMBLE THE BOOT
RK1:RK=SJ,SYSDEV,SYCND,RK     !AND RK DRIVER
RK1:SYSTBL=SJ,SYCND,SYSTBL    !AND SYSTBL
^C
!ALL DONE
```

4.3.2 Executing Indirect Files

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

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

```
@filespec
```

where

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

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

```
.@INDCT
```

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

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

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

```
*.BAK
```

This is the command that will actually execute:

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

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

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

If device DX1: becomes full before all the files from DX0: are copied to it, the second line of the indirect file does not execute completely. Execution then passes to the next line and the system deletes all MACRO files from DX0:. The ^C returns control to the monitor, which aborts the rest of the indirect file. This example shows that it is possible to destroy files accidentally because of the way indirect files execute. To be safe, use only keyboard monitor commands in an indirect file. This way the monitor gets control after each operation and can abort the indirect file as soon as it detects an error. A better way to perform the same operations as the indirect file shown above is as follows:

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

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

Normally, as each line of an indirect file executes, it echoes on the console terminal so that you can observe the progress of the job. However, you can use the SET TT QUIET command, described in Section 4.4, to suppress this printout. In this case, only the prompting messages, if any, print. You can stop execution of an indirect file at any time by typing two CTRL/C characters. Control returns to the monitor and you can enter a new command. You can also abort the indirect file by typing a single CTRL/C in response to a query or prompt. If you use an indirect file to execute a MACRO program, read Section 2.4.15 of the *RT-11 Advanced Programmer's Guide* to learn about certain restrictions on using the .EXIT call with indirect files.

You can call another indirect file from within an indirect file. This procedure is called nesting. Restrict nesting to three levels of indirect files. The following example shows two-level nesting. Assume a programmer types this command at the console terminal in response to the monitor's prompt:

```
@FIRST
```

The file FIRST.COM contains these lines:

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

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

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

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

4.3.3 Startup Indirect Files

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

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

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

4.4 KEYBOARD MONITOR COMMANDS

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

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

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

If you make a mistake in a command line, or if the system cannot perform the action you request, an error message prints on your terminal. The error message indicates which error occurred; see the *RT-11 System Message Manual* for a more complete description of the error and for the recommended action you should take. The error message also indicates which system utility program detected the error. This is for your information only and requires no action.

The APL command invokes the APL interpreter.

```
APL
```

APL has its own command language. Therefore, the APL command accepts no options and no file specifications.

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

ASSIGN (SP) physical-device-name (SP) logical-device-name

In the command syntax illustrated above, physical-device-name represents the RT-11 standard permanent name that refers to a particular device. Table 3-1 contains a list of these names. The term logical-device-name represents an alphanumeric name, from one to three characters long, that you assign to a particular device. Note that you should not use spaces or tabs in the logical device name. If you omit the physical device name, the system prompts you with Physical device name?. If you omit the logical device name, the system prompts you with Logical device name?.

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

If the logical device name you supply is already associated with a physical device, the system disassociates the logical name from that physical device and assigns it to the current device. You can assign only one logical name with each ASSIGN command, but you can use several ASSIGN commands to assign different logical names to the same device. You can also use the ASSIGN command to assign FORTRAN logical units to physical devices.

If you are running under the foreground/background monitor (FB), FB is not allowed as a logical device name. However, it is valid under the single-job monitor. Note that the following names are always illegal logical device names: BA, FG, and EL.

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

```
.ASSIGN LP: OUT:
```

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

The following command redirects printer output to the terminal.

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

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

The next command redefines the default file device.

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

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

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

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

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

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

B[(SP) address]

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

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

The following command sets the base to 0.

```
.B
```

The next two commands both set the base to 1000.

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

The BASIC command invokes the BASIC language interpreter.

BASIC

BASIC has its own command language. Therefore, the BASIC command accepts no options and no file specifications.

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

BOOT (SP) filespec

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

To perform a hardware bootstrap, specify only a device name in the command line. The following devices are legal for this operation: DT0:, RK0:-RK7:, RF:, SY:, DK:, DP0:-DP7:, DX0:-DX1:, DM0:-DM7:, and DS0:-DS7:. The hardware bootstrap operation gives control of the system to the particular monitor whose bootstrap is written on the device. (You can change this monitor by using the COPY/BOOT command.) This example bootstraps the single-job monitor, RKMNSJ, whose bootstrap information is written on device DK:.

```
.BOOT DK:

RT-11SJ  V03-01
```

To bootstrap a particular monitor file, specify that file name and the device on which it is stored, if necessary, in the command line. SY: is the default device and .SYS is the default file type. Note that the first two characters of the physical device name and the monitor file name must be the same, as in the following example.

```
.BOOT DX0:DXMNSJ

RT-11SJ  V03-01
```

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

```
.BOOT RKMNFB

RT-11FB  V03-01
```

The system recognizes only the RT-11 standard monitor names. You cannot, therefore, bootstrap a monitor file that has been given a non-standard name.

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

CLOSE

The CLOSE command accepts no options or arguments.

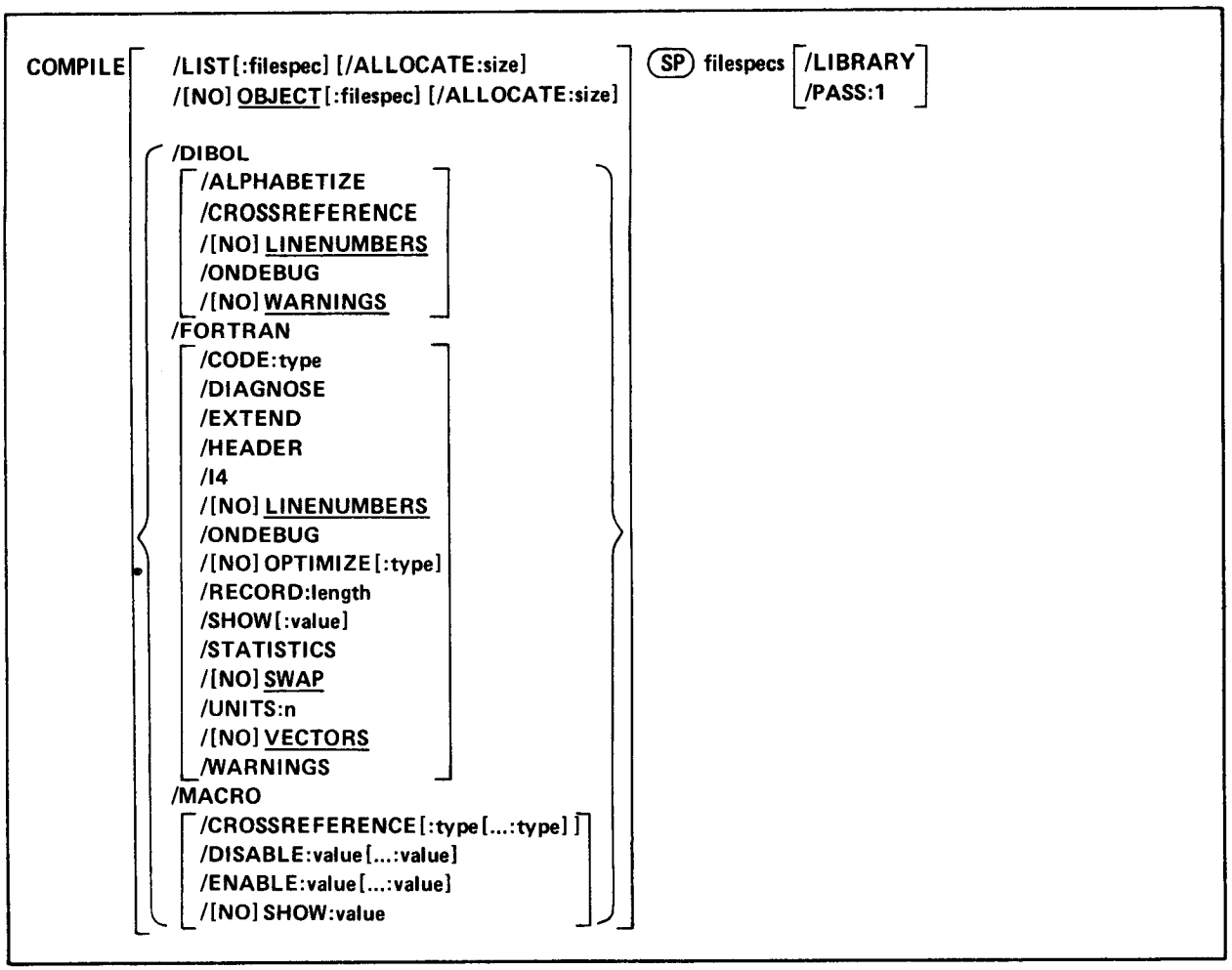
You can use the CLOSE command to make tentative open files permanent; otherwise, they do not appear in a normal directory listing and the space associated with the files is available for reuse. The CLOSE command is particularly useful after you type a CTRL/C to abort a background job. You can also use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

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

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

```
.R PROG
    .
    .
    .
^C^C
.CLOSE
```

The COMPILE command invokes one or more language processors to assemble or compile the files you specify.



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

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

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

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

There are several ways to establish which language processor the COMPILE command invokes. One way is to specify a language-name option, such as /MACRO, which invokes the MACRO assembler. Another way is to omit the

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

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

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

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

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

/CODE:type – Use this option with FORTRAN to produce object code that is designed for a particular hardware configuration. The argument, type, represents a three-letter abbreviation for the type of code to produce. The legal values are the following: EAE, EIS, FIS, and THR. See Section 1.1.1 of the *RT-11/RSTS/E FORTRAN IV User's Guide* for a complete description of the types of code and their functions.

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

With MACRO, this option takes an optional argument. The argument, type, represents a one-character code that indicates which sections of the cross-reference listing the assembler should include. Table 4-10 summarizes the valid arguments and their meaning.

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

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

/DISABLE:value[. . . :value] – Use this option with MACRO to specify a .DSABL directive. Table 4-11 summarizes the arguments and their meaning. See Section 6.2 of the *PDP-11 MACRO Language Reference Manual* for a description of the directive and a list of all legal values.

/ENABLE:value[. . . :value] – Use this option with MACRO to specify an .ENABL directive. Table 4-11 summarizes the arguments and their meaning. See Section 6.2 of the *PDP-11 MACRO Language Reference Manual* for a description of the directive and a list of all legal values.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

You cannot use a command line like the next one. In this example, the second listing file would replace the first one and, therefore, cause an error.

```
.COMPILE/LIST:FILE2 A.MAC,B.MAC
```

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

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

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

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

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

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

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

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

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

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

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

```
.COMPILE/FORTRAN A

.COMPILE/FORTRAN/OBJECT A
```

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

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

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

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

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

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

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

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

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

/OPTIMIZE[:type] – Use this option with FORTRAN to enable certain options that optimize object code for various conditions. The argument, type, represents the three-letter code for the type of optimization to enable. Table 4-4 summarizes the codes and their meanings.

/NOOPTIMIZE[:type] – Use this option with FORTRAN to disable certain options that optimize object code for various conditions. The argument, type, represents the three-letter code for the type of optimization to disable. Table 4-4 summarizes the codes and their meanings.

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

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

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

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

Use this option with MACRO to specify any MACRO .LIST directive. Table 4-12 summarizes the valid arguments and their meanings. Section 6.1.1 of the *PDP-11 MACRO Language Reference Manual* explains how to use these directives.

/NOSHOW:value – Use this option with MACRO to specify any MACRO .NLIST directive. Table 4-12 summarizes the valid arguments and their meanings. Section 6.1.1 of the *PDP-11 MACRO Language Reference Manual* explains how to use these directives.

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

/SWAP – Use this option with FORTRAN to permit the USR (user service routine) to swap over the FORTRAN program in memory. This is the default operation.

/NOSWAP – Use this option with FORTRAN to keep the USR resident during execution of a FORTRAN program. This may be necessary if the FORTRAN program uses some of the RT-11 System Subroutine Library calls (see Chapter 4 of the *RT-11 Advanced Programmer's Guide*). If the program frequently updates or creates a large number of different files, making the USR resident can improve program execution. However, the penalty for making the USR resident is 2K words of memory.

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

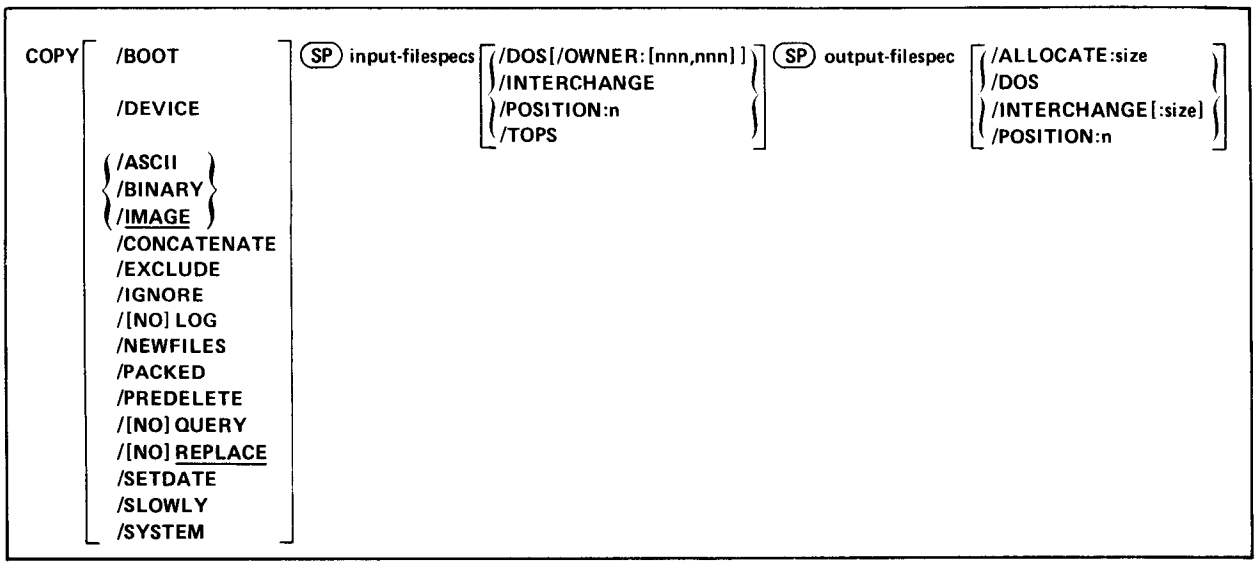
/VECTORS – This option directs FORTRAN to use tables to access multidimensional arrays. This is the default mode of operation.

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

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

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

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



The COPY command transfers:

- One file to another file
- A number of files to a single file by concatenation
- One device to another device
- A bootstrap to a device.

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

Normally, commas separate the input files if you specify more than one. However, you can separate them by plus (+) signs if you want to combine them. In this case, you can also omit the /CONCATENATE option, as the following example shows.

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

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

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

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

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

You can enter the COPY command as one line, or you can rely on the system to prompt you for information. The COPY command prompts are: From? for the input file specification and To? for the output file specification.

The system has a special way of handling system (.SYS) files and files that cover bad blocks (.BAD files). So that you do not copy system files by accident when you use a wildcard in the file specification, the system requires you to use the /SYSTEM option when you need to copy system files. To copy a .BAD file, you must specify it by explicitly giving its file name and file type. Since .BAD files cover bad blocks on a device, you usually do not need to copy, delete, or otherwise manipulate these files.

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

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

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

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

/BINARY – Use this option to copy formatted binary files. These include .OBJ files produced by the assembler or the FORTRAN compiler, and .LDA files produced by the linker. The system verifies checksums and prints a warning if a checksum error occurs. If this happens, the copy operation does not complete. Note that you cannot copy library files with the /BINARY option because of a checksum error. Copy them in image mode. The following command copies a binary file from DK: to a diskette.

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

/BOOT – This option copies bootstrap information from a monitor file to blocks 0 and 2 through 5 of a random access device. This permits you to use that device as a system device. Note that you cannot combine /BOOT with any other option. Before you use the /BOOT option, make sure that the appropriate monitor file is already stored on the disk. To create a bootable system diskette, for example, you could use the foreground/background file called DXMNFB.SYS. If you copy the monitor file onto the diskette from another device, be careful not to rename it. The COPY/BOOT operation recognizes only standard RT-11 monitor file names. You can use a procedure similar to the following to create a system device:

1. Initialize the disk. Use the monitor INITIALIZE command to do this.
2. Copy files onto the disk. Use the COPY/SYSTEM command for this step.
3. Use COPY/BOOT to write the monitor bootstrap onto the disk.

The following example shows how to create a system diskette.

```
.INITIALIZE DX1:

DX1:/Init are you sure?Y

.COPY/SYSTEM DX0:*.* DX1:*.*
 Files copied:
DX0:DXMNSJ.SYS to DX1:DXMNSJ.SYS
DX0:DT.SYS     to DX1:DT.SYS
DX0:DX.SYS     to DX1:DX.SYS
DX0:TT.SYS     to DX1:TT.SYS
DX0:LP.SYS     to DX1:LP.SYS
DX0:DIR.SAV    to DX1:DIR.SAV
DX0:DUP.SAV    to DX1:DUP.SAV
```

```
DX0:ABC.MAC      to DX1:ABC.MAC
DX0:AAF.MAC      to DX1:AAF.MAC
DX0:CT.SYS       to DX1:CT.SYS
DX0:PIP.SAV      to DX1:PIP.SAV
DX0:MT.SYS       to DX1:MT.SYS
DX0:MM.SYS       to DX1:MM.SYS
DX0:COMB.        to DX1:COMB.
DX0:DXMNFB.SYS   to DX1:DXMNFB.SYS

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

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

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

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

If one device is smaller than the other, the system copies only as many blocks as the smaller device contains. It is possible to copy blocks between disk and magtape, even though magtape is not a random access device. The data is stored on tape formatted in 1K word blocks. There is room for only one disk image on a magtape. The following command copies an image of DX0: to DX1:.

```
.COPY/DEVICE DX0: DX1:

DX1:/Copy are you sure?Y
```

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

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

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

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

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

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

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

/IGNORE – Use this option to ignore input errors during a copy operation. /IGNORE forces a single-block data transfer, which you can invoke at any other time with the /SLOWLY option. Use /IGNORE if an input error occurred when you tried to perform a normal copy operation. This procedure can sometimes recover a file that is otherwise unreadable. If there is still an error, an error message prints on the terminal, but the copy operation continues. This option is illegal with /DOS, /TOPS, and /INTERCHANGE.

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

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

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

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

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

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

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

```
.COPY DX1:*.SAV DX0:*.SAV
 Files copied:
DX1:DIR.SAV     to DX0:DIR.SAV
DX1:DUP.SAV     to DX0:DUP.SAV
DX1:PIP.SAV     to DX0:PIP.SAV
```

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

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

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

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

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

/POSITION:n – Use this option when you copy files to or from magtape or cassette. The /POSITION:n option lets you direct the tape operation; you can move the tape and perform an operation at the point you specify. For all operations, omitting the argument, n, has the same effect as setting n equal to 0 (n is interpreted as a decimal number).

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

1. If n is 0:
   The tape rewinds and the system searches for the file you specify. If you specify more than one file, the tape rewinds before each search. If the file specification contains a wildcard, the tape rewinds only once and then the system copies all the appropriate files.
2. If n is a positive integer:
   The system looks for the file at file sequence number n. If the file it finds there is the one you specify, the system copies it. Otherwise, the system prints an error message. If you use a wildcard in the file specification, the system goes to file sequence number n and then begins to look for the appropriate files.
3. If n is -1:
   The system starts its search at the current position. Note that if the current position is not the beginning of the tape, it is possible that the file you specify will not be found, even though it does exist on the tape.

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

1. If n is 0:
   The tape rewinds before the system copies each file. A warning message prints on the terminal if the system finds another file on the tape with the same name and file type.
2. If n is a positive integer:
   The system goes to file sequence number n or to the logical end of tape, whichever comes first. Then it enters the file you specify. If you specify more than one file, or if you use a wildcard in the file specification, the tape does not rewind before the system writes each file, and the system does not check for duplicate file names.
3. If n is -1:
   The system goes to the logical end of tape and enters the file you specify. It does not check for duplicate file names.
4. If n is -2:
   The tape rewinds between each copy operation. The system enters the file you specify at logical end-of-tape or at the first occurrence of a duplicate file name.

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

1. If n is 0:
   The cassette rewinds and the system searches for the file you specify. If you specify more than one file, or if you use a wildcard in the file specification, the cassette rewinds before each search.
2. If n is a positive integer:
   The system starts from the cassette's present position and searches for the file you specify. If the system does not find the file you specify before it reaches the nth file from its starting position, it reads the nth file. Note that if the starting position is not the beginning of the tape, it is possible that the system will not find the file you specify, even though it does exist on the tape.

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

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

1. If n is 0:
   The cassette rewinds and the system writes the file you specify at the logical end-of-tape. The system automatically deletes any file it finds along the way that has the same name and file type as the file you specify.
2. If n is a positive integer:
   The system starts from the cassette's present position and searches n files ahead, deleting along the way any file it finds that has the same name and file type as the file you specify. If the system does not reach the logical end-of-tape before it reaches the nth file from its starting position, it enters the file you specify over the nth file and deletes any files beyond it on the tape. If the system reaches the logical end-of-tape before it reaches the nth file, it writes the file you specify at the end-of-tape position.
3. If n is a negative integer:
   The cassette rewinds, then the system follows the same procedure outlined in step 2 above.

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

/PREDELETE – This option deletes a file on the output device if you copy a file with the same name to that device. The system deletes the file on the output device before the copy occurs. Normally, the system deletes a file of the same name after the copy operation successfully completes. This option is useful for operations involving devices that have limited space, such as diskette. Be careful when you use the /PREDELETE option; if for any reason the input file is unreadable, the output file will already have been deleted and you can be left with no useable version of the file.

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

```
.COPY/QUERY DK:*.FOR DX1:*.*
 Files copied:
DK:A.FOR        to DX1:A.FOR       ? Y
DK:B.FOR        to DX1:B.FOR       ? Y
DK:C.FOR        to DX1:C.FOR       ? NO
DK:DEMOF1.FOR   to DX1:DEMOF1.FOR? Y
```

/NOQUERY – This option suppresses the confirmation message that the system prints for some operations, such as COPY/DEVICE.

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

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

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

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

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

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

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

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

D (SP) address=value[, . . . value]

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

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

Use commas to separate multiple values in the command line. Two or more adjacent commas cause the system to deposit 0s at the location you specify and at the following locations, if indicated.

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

The following command deposits 0s into locations 300, 302, 304, and 306.

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

The next command sets the base address to 0.

```
.B
```

The following command deposits 3705 into location 1000.

```
.D 1000=3705
```

The next command sets the relocation base to 1000.

```
.B 1000
```

The last command puts 2503 into location 1500 and 22 into location 1502.

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

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

```
DATE[ (SP) dd-mmm-yy]
```

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

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

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

The system automatically changes the date each day at midnight. However, it does not change the date correctly at the end of each month. Do this by issuing the DATE command.

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

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

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

DEASSIGN[(SP) logical-device-name]

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

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

```
.DEASSIGN INP:
```

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

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

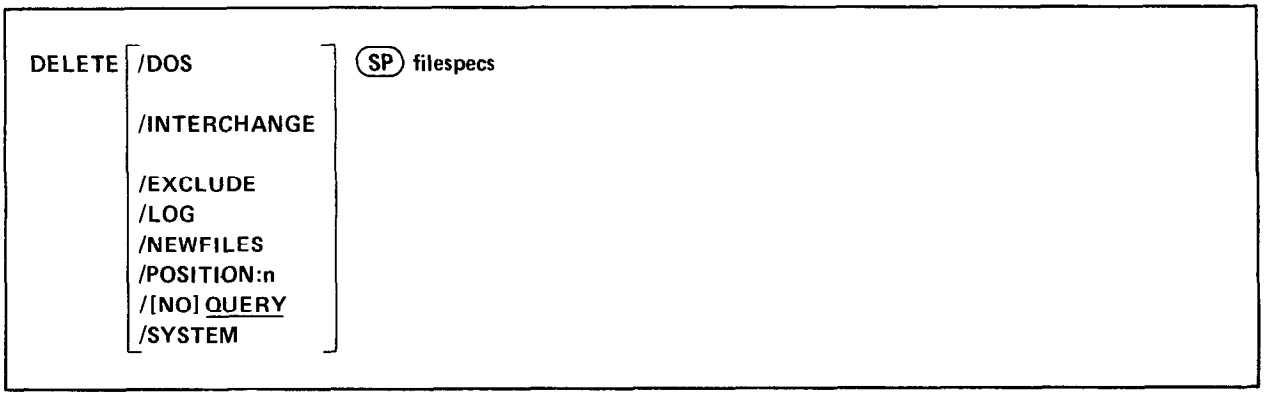
To disassociate all logical names from physical devices, type the DEASSIGN command without an argument. The following example disassociates all logical device names (except DK: and SY:) from physical devices.

```
.DEASSIGN
```

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

```
.DEASSIGN DK:
```

The DELETE command deletes the files you specify.



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

The system has a special way of handling system (.SYS) files and files that cover bad blocks (.BAD files). So that you do not delete system files by accident when you use a wildcard in the file specification, the system requires you to use the /SYSTEM option when you need to delete system files. To delete a .BAD file, you must specify it by explicitly giving its file name and file type. Since .BAD files cover bad blocks on a device, you usually do not need to copy, delete, or otherwise manipulate these files.

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

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

/DOS – Use this option to delete a file that is in DOS-11 or RSTS/E format. Remember that the valid devices for this type of file are disks and DECtape. You cannot combine any other option with /DOS.

/EXCLUDE – This option deletes all the files on a device except the ones you specify. The following command, for example, deletes all files from DX1: except .SAV files. Remember to use /SYSTEM if you need to include .SYS files in the operation.

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

/INTERCHANGE – Use this option to delete from a diskette a file that is in interchange (proposed ANSI standard) format. You cannot combine any other option with /INTERCHANGE.

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

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

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

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

1. If n is 0:
   The cassette rewinds and the system searches for the file you specify. If you specify more than one file, or if you use a wildcard in the file specification, the cassette rewinds before each search.
2. If n is a positive integer:
   The system starts from the cassette's present position and searches for the file you specify. If the system does not find the file you specify before it reaches the nth file from its starting position, it deletes the nth file. Note that if the starting position is not the beginning of the tape, it is possible that the system will not find the file you specify, even though it does exist on the tape.
3. If n is a negative integer:
   The cassette rewinds, then the system follows the procedure outlined in step 2 above.

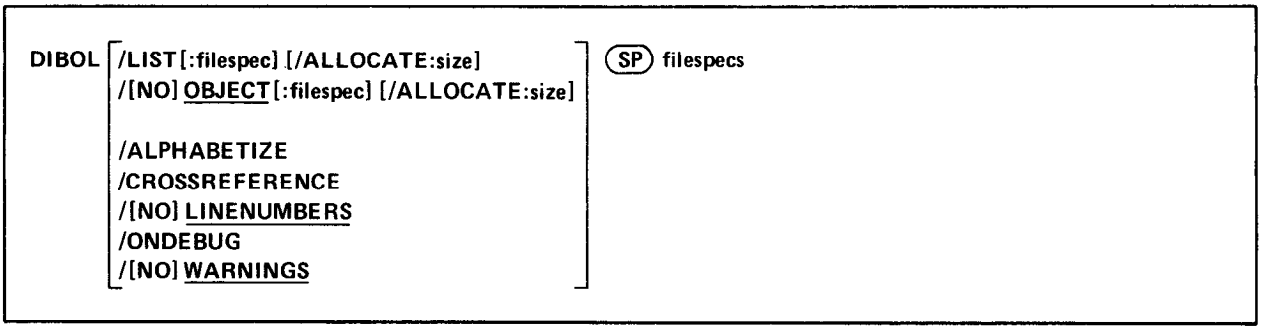
/QUERY – Use this option to request a confirmation message from the system before it deletes each file. This option is particularly useful on operations that involve wildcards, when you may not be completely sure which files the system selected for the operation. This is the default mode of operation.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

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

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

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

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



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

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

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

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

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

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

/CROSSREFERENCE – This option generates a symbol cross-reference section in the listing. This options adds as many as four separate sections to the listing. These sections are: 1) symbol cross-reference table, 2) label cross-reference table, 3) external subroutine cross-reference table, 4) COMMON cross-reference table. Note that the system does not generate a listing by default. You must also specify /LIST in the command line to get a cross-reference listing.

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

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

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

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

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

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

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

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

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

You cannot use a command line like the next one. In this example, the second listing file would replace the first one and, therefore, cause an error.

```
.DIBOL/LIST:FILE2 A,B
```

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

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

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

```
.DIBOL A/LIST:B

.DIBOL/LIST:B A
```

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

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

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

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

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

```
.DIBOL A

.DIBOL/OBJECT A
```

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

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

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

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

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

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

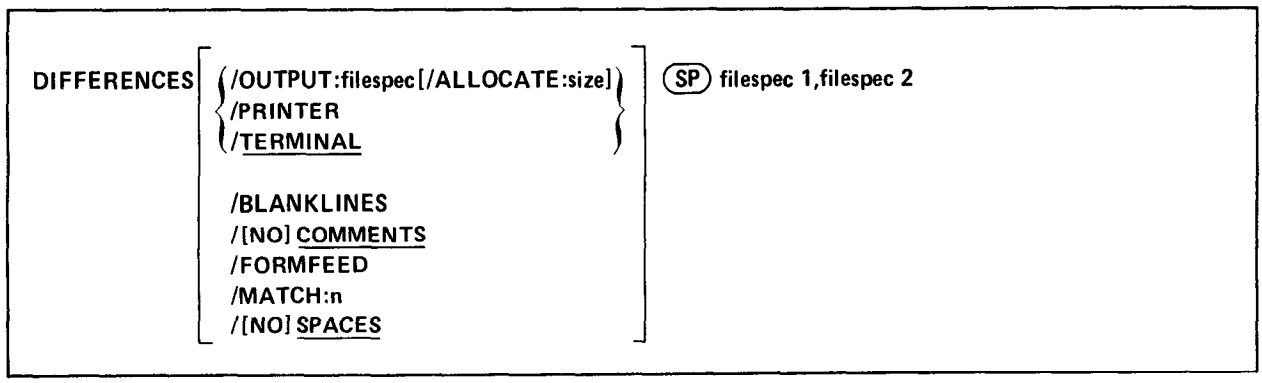
```
.DIBOL A+B/LIST,C/NOOBJECT/LIST
```

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

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

/NOWARNINGS – Use this option to suppress warning messages during compilation. These messages are for your information only; they do not affect the compilation.

The DIFFERENCES command compares two files and lists the differences between them in a file or on a device.



In the command syntax shown above, filespec1 represents the first file to be compared and filespec2 represents the second file to be compared. The default output device is the console terminal. The default file type for input files is .MAC; for output files it is .DIF. You can specify the entire command on one line, or you can rely on the system to prompt you for information. The DIFFERENCES command prompts are File 1? and File 2?.

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

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

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

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

/NOCOMMENTS – Use this option to exclude comments (text on a line preceded by a semicolon) and spacing (spaces and tabs) from the comparison. This is useful if you are comparing two MACRO source programs with similar contents but different formats.

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

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

/OUTPUT:filespec – Use this option to specify a device and file name for the output listing file. Normally, the listing appears on the console terminal. If you omit the file type for the listing file, the system uses .DIF.

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

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

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

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

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

```
.TYPE FILE1.TXT
        FILE1
HERE'S A BOTTLE AND AN HONEST FRIEND!
  WHAT WAD YE WISH FOR MAIR, MAN?
WHA KENS, BEFORE HIS LIFE MAY END,
  WHAT HIS SHAME MAY BE O' CARE, MAN?
THEN CATCH THE MOMENTS AS THEY FLY,
  AND USE THEM AS YE OUGHT, MAN: --
BELIEVE ME, HAPPINESS IS SLY,
  AND COMES NOT AY WHEN SOUGHT, MAN.

                          --SCOTTISH SONG
```

```
.TYPE FILE2.TXT
        FILE1
HERE'S A BOTTLE AND AN HONEST FRIEND!
  WHAT WAD YE WISH FOR MAIR, MAN?
WHA KENS, BEFORE HIS LIFE MAY END,
  WHAT HIS SHARE MAY BE O' CARE, MAN?
THEN CATCH THE MOMENTS AS THEY FLY,
  AND USE THEM AS YE OUGHT, MAN: --
BELIEVE ME, HAPPINESS IS SHY,
  AND COMES NOT AY WHEN SOUGHT, MAN.

                          --SCOTTISH SONG
```

Notice that FILE1.TXT contains two typing errors. In the fourth line of the song, "shame" should be "share." In the seventh line, "sly" should be "shy."

The following command compares the two files, creating a listing file called DIFF.TXT.

```
.DIFFERENCES/MATCH:1/OUTPUT:DIFF.TXT FILE1.TXT,FILE2.TXT

%FILES ARE DIFFERENT
```

The following listing shows file DIFF.TXT.

```
.TYPE DIFF.TXT
1)1             FILE1
2)1             FILE1
```

```
1)1        WHAT HIS SHAME MAY BE O' CARE, MAN?
1)       THEN CATCH THE MOMENTS AS THEY FLY,
****
2)1        WHAT HIS SHARE MAY BE O' CARE, MAN?
2)       THEN CATCH THE MOMENTS AS THEY FLY,
**********
1)1      BELIEVE ME, HAPPINESS IS SLY,
1)         AND COMES NOT AY WHEN SOUGHT, MAN.
****
2)1      BELIEVE ME, HAPPINESS IS SHY,
2)         AND COMES NOT AY WHEN SOUGHT, MAN.
**********
```

If the files are different, the system always prints the first line of each file as identification.

```
1)1              FILE1
2)1              FILE1
```

The numbers at the left margin have the form n)m, where n represents the source file (either 1 or 2) and m represents the page of that file on which the specific line is located.

The system next prints a blank line and then lists the differences between the two files. The /MATCH:n option was used in this example to set to 1 the number of lines that must agree to constitute a match.

The first three lines of the song are the same in both files, so they do not appear in the listing. The fourth line contains the first discrepancy. The system prints the fourth line from the first file, followed by the next matching line as a reference.

```
1)1          WHAT HIS SHAME MAY BE O' CARE, MAN?
1)         THEN CATCH THE MOMENTS AS THEY FLY,
****
```

The four asterisks terminate the differences section from the first file.

The system then prints the fourth line from the second file, again followed by the next matching line as a reference:

```
2)1          WHAT HIS SHARE MAY BE O' CARE, MAN?
2)         THEN CATCH THE MOMENTS AS THEY FLY,
**********
```

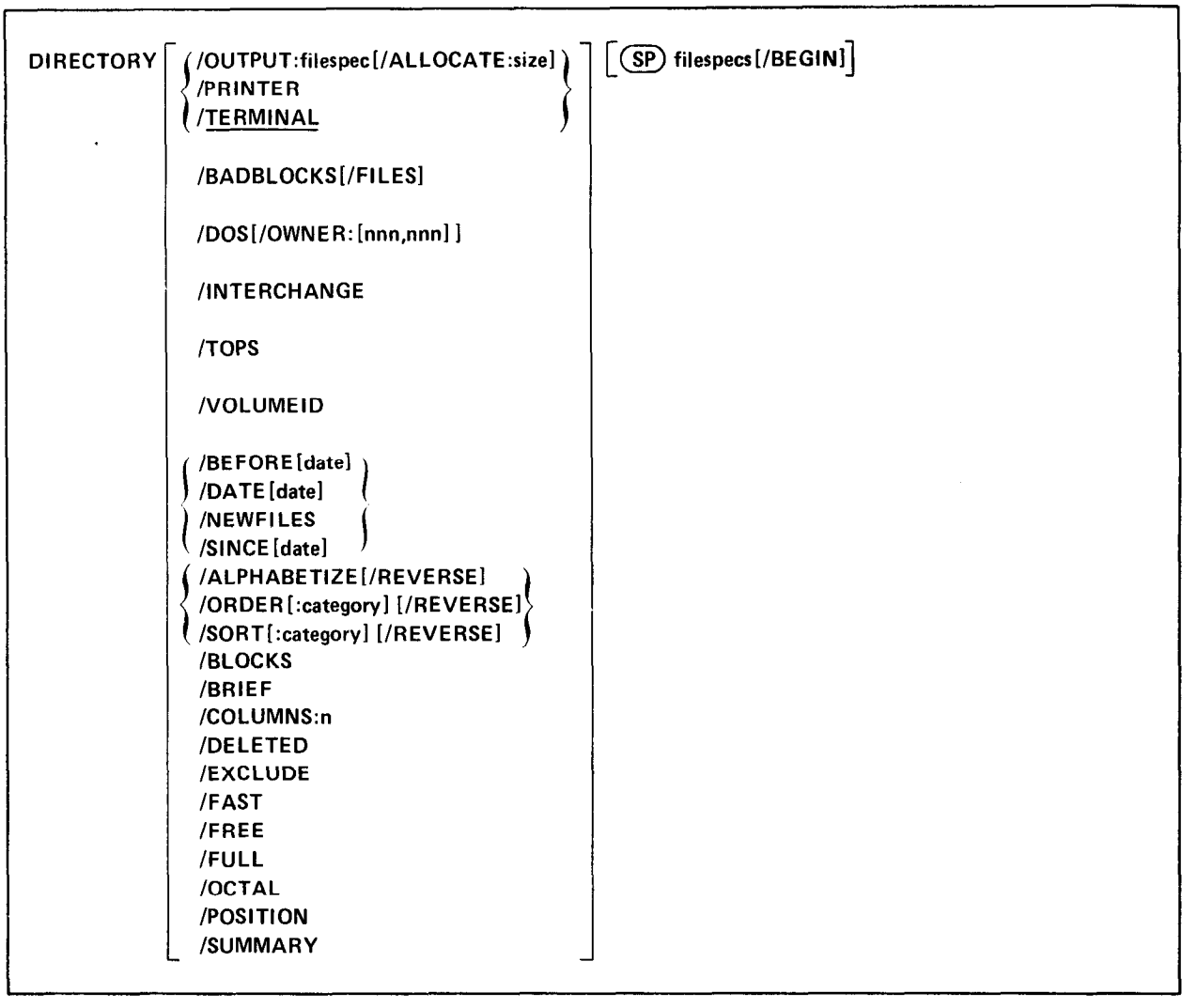
The ten asterisks terminate the listing for a particular difference section.

The system scans the remaining lines in the files in the same manner. When it reaches the end of each file, it prints the %FILES ARE DIFFERENT message on the terminal.

If you compare two files that are identical, the system does not create an output file or listing, as this example shows.

```
.DIFFERENCES FILE1.TXT,FILE1.TXT
NO DIFFERENCES ENCOUNTERED
```

The DIRECTORY command lists information you request about a device, a file, or a group of files.



In the command syntax shown above, filespecs represents the device, file, or group of files whose directory information you request. The DIRECTORY command can list directory information about a specific device, such as the number of files stored on the device, their names, and their creation dates. It can list details about certain files, too, including their names, their file types, and their size in blocks. You can specify up to six files explicitly, but you can obtain directory information about many files by using wildcards in the file specification. The DIRECTORY command can also print a device directory summary, and it can organize its listings in several ways, such as alphabetically or chronologically.

Normally, the DIRECTORY command prints listings in two columns on the terminal. Read these listings as you would read a book: read across the columns, moving from left to right, one row at a time. Directory listings that are sorted (with /ALPHABETIZE, /ORDER, or /SORT) are an exception to this. Read these listings by reading the left column from top to bottom, then reading the right column from top to bottom.

The DIRECTORY command does not prompt you for any information. If you omit the file specification, the system lists directory information about device DK:, as this example shows.

```
.DIRECTORY
 19-May-77
DXMNSJ.SYS     88 08-Apr-77     AAF    .MAC      2 19-Apr-77
FIX463.SAV      2 29-Jul-76     ABC    .MAC      4 19-Apr-77
JMUL   .OBJ     1 03-May-77     DEMOFG.MAC       5 18-Jan-77
PTCH   .BAK     1 05-May-77     CT     .SYS      5 08-Apr-77
DX     .SYS     3 08-Apr-77     MERGE  .FOR      6 24-Apr-77
MYPROG.MAC      7 24-Feb-77     VTMAC  .MAC      7 31-Aug-76
ALIB   .OBJ     3 03-May-77     MX     .SYS      9 08-Apr-77
DXMNFB.SYS     97 08-Apr-77     DIR    .SAV     16 08-Apr-77
DUP    .SAV    17 13-Apr-77     PIP    .SAV     16 14-Apr-77
 18 Files, 289 Blocks
 191 Free blocks
```

If you specify only a device in the file specification, the system lists directory information about all the files on that device. If you specify a file name, the system lists information about just that file, as this example shows.

```
.DIRECTORY DX0:MYPROG.MAC
 19-May-77
MYPROG.MAC      7 24-Feb-77
 1 Files, 7 Blocks
 191 Free blocks
```

The following sections describe the options you can use with the DIRECTORY command and provide sample directory listings. Some of the options accept a date or part of a date as an argument. The syntax for specifying the date is:

[:dd] [:mmm] [:yy]

where

dd represents the day (a decimal integer in the range 1-31).

mmm represents the first three characters of the name of the month.

yy represents the year (a decimal integer in the range 73-99).

The default value for the date is the current system date. If you specify just the day, the system interprets it as the given day of the current month and year. If you specify just the month, the system interprets it as the first day of the given month in the current year. If you specify only the year, the system interprets it as the start of that year. If the current system date is not set, it is considered 0 (the same as for an undated file in a directory listing).

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

/ALPHABETIZE – This option lists the directory of the device you specify in alphabetical order by file name and file type. It has the same effect as the /ORDER:NAME option.

/BADBLOCKS – Sometimes devices (disks and DECtapes) are manufactured with bad blocks, or they develop bad blocks as a result of use and age. Use the /BADBLOCKS option to scan a device and locate bad blocks on it. The system prints the absolute block number of these blocks on the devices that return hardware errors when

the system tries to read them. This procedure does not destroy data that is already stored on the device. Remember that block numbers are octal and the first block on a device is block 0. If a device has no bad blocks, only the heading prints, as this example shows.

```
.DIRECTORY/BADBLOCKS DX1:
BAD BLOCKS   TYPE   FILENAME      REL BLK
```

/BEFORE[date] – This option prints a directory of files created before the date you specify. The following command lists on the terminal all files stored on device DX0: that were created before April 1977.

```
.DIRECTORY/BEFORE:APR DX0:
 24-May-77
FIX463.SAV      2 29-Jul-76     DEMOFG.MAC     5 18-Jan-77
MYPROG.MAC      7 24-Feb-77     VTMAC .MAC     7 31-Aug-76
 4 Files, 21 Blocks
 191 Free blocks
```

/BEGIN – This option lists the directory of the device you specify, beginning with the file you name and including all the files that follow it in the directory. The occurrence of file names in the listing is the same as the order of the files on the device.

The following example lists the file VTMAC.MAC on device DX0: and all the files that follow it in the directory.

```
.DIRECTORY DX0:VTMAC.MAC/BEGIN
 24-May-77
VTMAC .MAC      7 31-Aug-76     ALIB  .OBJ     3 03-May-77
MX    .SYS      9 08-Apr-77     DXMNFB.SYS    97 08-Apr-77
DIR   .SAV     16 08-Apr-77     DUP   .SAV    17 13-Apr-77
PIP   .SAV     16 14-Apr-77
 7 Files, 165 Blocks
 191 Free blocks
```

/BLOCKS – This option prints a directory of the device you specify and includes the starting block number in decimal of all the files listed. The following example lists the directory of DX0:, including the starting block numbers of files.

```
.DIRECTORY/BLOCKS DX0:
 19-May-77
DXMNSJ.SYS   88 08-Apr-77   14    AAF   .MAC     2 19-Apr-77   102
FIX463.SAV    2 29-Jul-76  104    ABC   .MAC     4 19-Apr-77   106
JMUL  .OBJ    1 03-May-77  110    DEMOFG.MAC     5 18-Jan-77   138
PTCH  .BAK    1 05-May-77  143    CT    .SYS     5 08-Apr-77   150
DX    .SYS    3 08-Apr-77  155    MERGE .FOR     6 24-Apr-77   158
MYPROG.MAC    7 24-Feb-77  164    VTMAC .MAC     7 31-Aug-76   171
ALIB  .OBJ    3 03-May-77  178    MX    .SYS     9 08-Apr-77   189
DXMNFB.SYS   97 08-Apr-77  207    DIR   .SAV    16 08-Apr-77   327
DUP   .SAV   17 13-Apr-77  343    PIP   .SAV    16 14-Apr-77   360
 18 Files, 289 Blocks
 191 Free blocks
```

/BRIEF – This option lists only file names and file types, omitting file lengths and associated dates. It produces a 5-column listing, as the following example shows.

```
.DIRECTORY/BRIEF DX0:
 19-May-77
DXMNSJ.SYS     AAF    .MAC     FIX463.SAV     ABC    .MAC   JMUL   .OBJ
DEMOFG.MAC     PTCH   .BAK     CT     .SYS    DX     .SYS   MERGE  .FOR
MYPROG.MAC     VTMAC  .MAC     ALIB   .OBJ    MX     .SYS   DXMNFB.SYS
DIR    .SAV    DUP    .SAV     PIP    .SAV
 18 Files, 289 Blocks
 191 Free blocks
```

/COLUMNS:n – Use this option to list a directory in a specific number of columns. The value, n, represents an integer in the range 1-9. Normally, the system uses two columns for regular listings and five columns for brief listings. The following example lists the directory information for device MT0: in one column.

```
.DIRECTORY/COLUMNS:1/POSITION MT0:
 15-Apr-77
VTMAC .MAC      7 15-Apr-77      1
SYCND .MAC      5 15-Apr-77      2
DIRECT.MAC    112 15-Apr-77      3
PIPSYM.MAC      4 15-Apr-77      4
PIP005.MAC    176 15-Apr-77      5
VTMAC .MAC      7 15-Apr-77      6
SYCND .MAC      5 15-Apr-77      7
DIRECT.MAC    112 15-Apr-77      8
PIPSYM.MAC      4 15-Apr-77      9
PIP005.MAC    176 15-Apr-77     10
 10 Files, 608 Blocks
```

In the example shown above, the numbers in the rightmost column represent the magtape file sequence numbers, which appear because of the /POSITION option.

/DATE[date] – Use this option to include in the directory listing only those files with the date you specify. The following command lists all the files on device DX0: that were created on 8 April 1977.

```
.DIRECTORY/DATE:8:APR:77 DX0:
 19-May-77
DXMNSJ.SYS     88 08-Apr-77     CT     .SYS      5 08-Apr-77
DX     .SYS     3 08-Apr-77     MX     .SYS      9 08-Apr-77
DXMNFB.SYS     97 08-Apr-77     DIR    .SAV     16 08-Apr-77
 6 Files, 218 Blocks
 191 Free blocks
```

/DELETED – This option lists a directory of the device you specify, listing the file names, types, sizes, creation dates and starting block numbers in decimal of files that have been deleted but whose file name information has not been destroyed. The file names that print represent either tentative files or files that have been deleted. This can be useful in recovering files that have been accidentally deleted. Once you identify the file name and location, you can use DUP to rename the area. See Section 8.2.1 for this procedure. The following command lists files on device DT1: that have been deleted.

```
.DIRECTORY/DELETED DT1:
 19-May-77
TEST   .LST   530 27-Apr-77     48
 0 Files, 0 Blocks
 0 Free blocks
```

Note in the example shown above that, since a deleted file does not really exist, the total number of files, blocks, and free blocks is 0.

/DOS – Use this option to list the directory of a device that is in RSTS/E or DOS/BATCH format. The only other options valid with /DOS are /BRIEF, /FAST, and /OWNER. The valid devices are DECtape and RK05.

/EXCLUDE – This option lists a directory of all the files on a device except those files you specify. The following example lists all files on DX0: except the .SAV and .SYS files.

```
.DIRECTORY/EXCLUDE DX0:(*.SAV,*.SYS)
 24-May-77
AAF    .MAC      2 19-Apr-77     ABC    .MAC      4 19-Apr-77
JMUL   .OBJ      1 03-May-77     DEMOFG.MAC      5 18-Jan-77
PTCH   .BAK      1 05-May-77     MERGE  .FOR      6 24-Apr-77
MYPROG.MAC       7 24-Feb-77     VTMAC  .MAC      7 31-Aug-76
ALIB   .OBJ      3 03-May-77
 9 Files, 36 Blocks
 191 Free blocks
```

/FAST – This option lists only file names and file types, omitting file lengths and associated dates. This option is the same as /BRIEF.

/FILES – Use this option with /BADBLOCKS to print the file names of bad blocks. This is particularly useful if the device is not a standard RT-11 directory-structured device. If the system does not find any bad blocks, it prints only the heading, as this example shows.

```
.DIRECTORY/BADBLOCKS/FILES DT1:
BAD BLOCKS   TYPE   FILENAME      REL BLK
```

/FREE – Use this option to print a directory of unused areas and their size. This example lists the unused areas on device DK:.

```
.DIRECTORY/FREE
 19-May-77
< UNUSED >      1          < UNUSED >      1
< UNUSED >      1          < UNUSED >      2
< UNUSED >      1          < UNUSED >      2
< UNUSED >     24          < UNUSED >     38
< UNUSED >     40          < UNUSED >      3
< UNUSED >      1          < UNUSED >      2
< UNUSED >      5          < UNUSED >      2
< UNUSED >     98
 0 Files, 0 Blocks
 221 Free blocks
```

/FULL – This option lists the entire directory, including unused areas and their sizes in blocks (decimal). The following example lists the entire directory for device DT0:.

```
.DIRECTORY/FULL DT0:
 19-May-77
EDIT1  .DEM      1 03-May-77     EDIT2  .DEM      1 03-May-77
FIX463.SAV       2 29-Jul-76     FILE1  .TXT      1 19-May-77
EDIT3  .DEM      1 03-May-77     FORTRA.SAV    201 01-May-77
PUTSTR.OBJ       7 14-Apr-77     PROMPT.KEP      2 05-May-77
```

```
PROMPT.SAV      2 05-May-77     ROOT   .SAV     3 05-May-77
ROOT   .KEP     3 05-May-77     PROMPT.BAK      2 05-May-77
PROMPT.MAC      2 05-May-77     PROMPT.OBJ      1 05-May-77
OVRLAY.BAK      1 05-May-77     PTCH   .BAK      1 05-May-77
PTCH   .MAC     1 05-May-77     OVRLAY.MAC      1 05-May-77
PTCH   .OBJ     1 05-May-77     OVRLAY.OBJ      1 05-May-77
FILE2  .TXT     1 19-May-77     < UNUSED >    328
 21 Files, 236 Blocks
 328 Free blocks
```

/INTERCHANGE – Use this option to list the directory of a diskette that is in interchange (proposed ANSI standard) format. The only other options valid with /INTERCHANGE are /BRIEF and /FAST.

/NEWFILES – This option includes in the directory listing only those files that were created today. This is a convenient way to list the files you created in a session at the computer. The following command lists the new files on 19 May 1977

```
.DIRECTORY/NEWFILES DT0:
 19-May-77
FILE1  .TXT     1 19-May-77     FILE2  .TXT     1 19-May-77
 2 Files, 2 Blocks
 328 Free blocks
```

/OCTAL – This option lists the sizes (and starting block numbers if you also use /BLOCKS) in octal. If the device you specify is a magtape or cassette, the system prints the sequence numbers in octal. The following example shows an octal listing of device DX0:.

```
.DIRECTORY/OCTAL DX0:
 19-May-77
DXMNSJ.SYS    130 08-Apr-77     AAF    .MAC     2 19-Apr-77
FIX463.SAV      2 29-Jul-76     ABC    .MAC     4 19-Apr-77
JMUL   .OBJ     1 03-May-77     DEMOFG.MAC      5 18-Jan-77
PTCH   .BAK     1 05-May-77     CT     .SYS     5 08-Apr-77
DX     .SYS     3 08-Apr-77     MERGE  .FOR     6 24-Apr-77
MYPROG.MAC      7 24-Feb-77     VTMAC  .MAC     7 31-Aug-76
ALIB   .OBJ     3 03-May-77     MX     .SYS    11 08-Apr-77
DXMNFB.SYS    141 08-Apr-77     DIR    .SAV    20 08-Apr-77
DUP    .SAV    21 13-Apr-77     PIP    .SAV    20 14-Apr-77
 18 Files,    441 Blocks
    277 Free blocks
```

/ORDER[:category] – This option sorts the directory of a device according to the category you specify. Table 4-3 summarizes the categories and their functions.

Table 4-3 Sort Categories

Category	Explanation
DATE	Sorts the directory chronologically by creation date. Files that have the same date are sorted alphabetically by file name and file type
NAME	Sorts the directory alphabetically by file name. Files that have the same file name are sorted alphabetically by file type (this has the same effect as the /ALPHABETIZE option).
POSITION	Lists the files in order by their position on the device. This is the same as using /ORDER with no category.
SIZE	Sorts the directory based on file size in blocks. Files that are the same size are sorted alphabetically by file name and file type.
TYPE	Sorts the directory alphabetically by file type. Files that have the same file type are sorted alphabetically by file name.

The following examples list the directory of device DX0:, in order by each of the categories.

```
.DIRECTORY/ORDER:DATE DX0:
 19-May-77
FIX463.SAV      2 29-Jul-76     MX     .SYS     9 08-Apr-77
VTMAC .MAC      7 31-Aug-76     DUP    .SAV    17 13-Apr-77
DEMOFG.MAC      5 18-Jan-77     PIP    .SAV    16 14-Apr-77
MYPROG.MAC      7 24-Feb-77     AAF    .MAC     2 19-Apr-77
CT     .SYS     5 08-Apr-77     ABC    .MAC     4 19-Apr-77
DIR    .SAV    16 08-Apr-77     MERGE  .FOR     6 24-Apr-77
DX     .SYS     3 08-Apr-77     ALIB   .OBJ     3 03-May-77
DXMNFB.SYS     97 08-Apr-77     JMUL   .OBJ     1 03-May-77
DXMNSJ.SYS     88 08-Apr-77     PTCH   .BAK     1 05-May-77
 18 Files, 289 Blocks
 191 Free blocks


.DIRECTORY/ORDER:NAME DX0:
 19-May-77
AAF    .MAC     2 19-Apr-77     DXMNSJ.SYS    88 08-Apr-77
ABC    .MAC     4 19-Apr-77     FIX463.SAV     2 29-Jul-76
ALIB   .OBJ     3 03-May-77     JMUL   .OBJ     1 03-May-77
CT     .SYS     5 08-Apr-77     MERGE  .FOR     6 24-Apr-77
DEMOFG.MAC      5 18-Jan-77     MX     .SYS     9 08-Apr-77
DIR    .SAV    16 08-Apr-77     MYPROG.MAC      7 24-Feb-77
DUP    .SAV    17 13-Apr-77     PIP    .SAV    16 14-Apr-77
DX     .SYS     3 08-Apr-77     PTCH   .BAK     1 05-May-77
DXMNFB.SYS     97 08-Apr-77     VTMAC  .MAC     7 31-Aug-76
 18 Files, 289 Blocks
 191 Free blocks


.DIRECTORY/ORDER:POSITION DX0:
 19-May-77
DXMNSJ.SYS     88 08-Apr-77     MERGE  .FOR     6 24-Apr-77
AAF    .MAC     2 19-Apr-77     MYPROG.MAC      7 24-Feb-77
FIX463.SAV      2 29-Jul-76     VTMAC  .MAC     7 31-Aug-76
ABC    .MAC     4 19-Apr-77     ALIB   .OBJ     3 03-May-77
JMUL   .OBJ     1 03-May-77     MX     .SYS     9 08-Apr-77
DEMOFG.MAC      5 18-Jan-77     DXMNFB.SYS     97 08-Apr-77
PTCH   .BAK     1 05-May-77     DIR    .SAV    16 08-Apr-77
CT     .SYS     5 08-Apr-77     DUP    .SAV    17 13-Apr-77
DX     .SYS     3 08-Apr-77     PIP    .SAV    16 14-Apr-77
 18 Files, 289 Blocks
 191 Free blocks


.DIRECTORY/ORDER:SIZE DX0:
 19-May-77
JMUL   .OBJ     1 03-May-77     MERGE  .FOR     6 24-Apr-77
PTCH   .BAK     1 05-May-77     MYPROG.MAC      7 24-Feb-77
AAF    .MAC     2 19-Apr-77     VTMAC  .MAC     7 31-Aug-76
FIX463.SAV      2 29-Jul-76     MX     .SYS     9 08-Apr-77
ALIB   .OBJ     3 03-May-77     DIR    .SAV    16 08-Apr-77
```

```
DX      .SYS       3 08-Apr-77     PIP     .SAV    16 14-Apr-77
ABC     .MAC       4 19-Apr-77     DUP     .SAV    17 13-Apr-77
CT      .SYS       5 08-Apr-77     DXMNSJ.SYS      88 08-Apr-77
DEMOFG.MAC         5 18-Jan-77     DXMNFB.SYS      97 08-Apr-77
 18 Files, 289 Blocks
 191 Free blocks


.DIRECTORY/ORDER:TYPE DX0:
 19-May-77
PTCH    .BAK       1 05-May-77     DIR     .SAV    16 08-Apr-77
MERGE   .FOR       6 24-Apr-77     DUP     .SAV    17 13-Apr-77
AAF     .MAC       2 19-Apr-77     FIX463.SAV       2 29-Jul-76
ABC     .MAC       4 19-Apr-77     PIP     .SAV    16 14-Apr-77
DEMOFG.MAC         5 18-Jan-77     CT      .SYS     5 08-Apr-77
MYPROG.MAC         7 24-Feb-77     DX      .SYS     3 08-Apr-77
VTMAC   .MAC       7 31-Aug-76     DXMNFB.SYS      97 08-Apr-77
ALIB    .OBJ       3 03-May-77     DXMNSJ.SYS      88 08-Apr-77
JMUL    .OBJ       1 03-May-77     MX      .SYS     9 08-Apr-77
 18 Files, 289 Blocks
 191 Free blocks
```

/OUTPUT:filespec – Use this option to specify a device and file name for the output listing file. Normally, the directory listing appears on the console terminal. If you omit the file type for the listing file, the system uses .DIR.

/OWNER:[nnn,nnn] – Use this option with /DOS to specify a user identification code (UIC). Note that the square brackets are part of the UIC; you must type them.

/POSITION – Use this option to list the file sequence numbers of files stored on a magtape. See /COLUMNS:n for a sample listing.

/PRINTER – Use this option to print the directory listing on the line printer. The default output device is the terminal.

/REVERSE – This option lists a directory in the reverse order of the sort you specify with /ALPHABETIZE, /ORDER, or /SORT. The following example sorts the directory of DX0: and lists it in reverse order by size.

```
.DIRECTORY/ORDER:SIZE/REVERSE DX0:
 24-May-77
DXMNFB.SYS      97 08-Apr-77     CT      .SYS     5 08-Apr-77
DXMNSJ.SYS      88 08-Apr-77     DEMOFG.MAC       5 18-Jan-77
DUP     .SAV    17 13-Apr-77     ABC     .MAC     4 19-Apr-77
DIR     .SAV    16 08-Apr-77     ALIB    .OBJ     3 03-May-77
PIP     .SAV    16 14-Apr-77     DX      .SYS     3 08-Apr-77
MX      .SYS     9 08-Apr-77     AAF     .MAC     2 19-Apr-77
MYPROG.MAC       7 24-Feb-77     FIX463.SAV       2 29-Jul-76
VTMAC   .MAC     7 31-Aug-76     JMUL    .OBJ     1 03-May-77
MERGE   .FOR     6 24-Apr-77     PTCH    .BAK     1 05-May-77
 18 Files, 289 Blocks
 191 Free blocks
```

/SINCE[date] – This option lists a directory of all files stored on the device you specify that were created on or after the date you specify. The following command lists only those files on DX0: that were created on or after 3 May 1977.

```
.DIRECTORY/SINCE:3:MAY:77 DX0:
 19-May-77
JMUL   .OBJ      1 03-May-77     PTCH   .BAK      1 05-May-77
ALIB   .OBJ      3 03-May-77
 3 Files, 5 Blocks
 191 Free blocks
```

/SORT[:category] – This option sorts the directory of a device according to the category you specify. This is the same as /ORDER[:category].

/SUMMARY – This option lists a summary of the segment structure of the device directory. The following example lists the segment structure of the directory for device DK:.

```
.DIRECTORY/SUMMARY
 19-May-77

   72 Files in segment 1

   46 Files in segment 2

   36 Files in segment 4

   33 Files in segment 3

   33 Files in segment 5

   16 Available segments, 5 in use

 220 Files, 4543 Blocks
 219 Free blocks
```

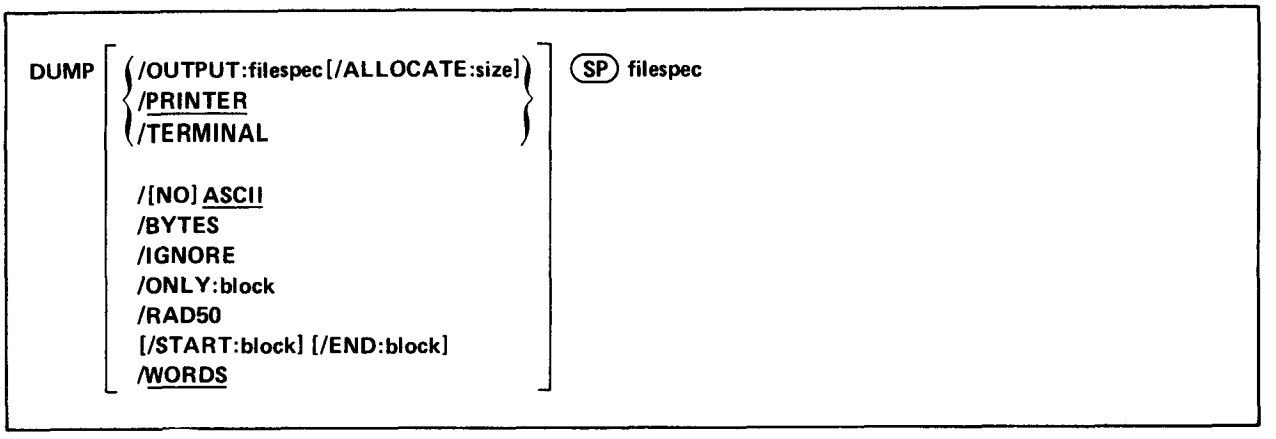
/TERMINAL – This option lists directory information on the console terminal. This is the default operation.

/TOPS – Use this option to list the directory of a DECtape that is in PDP-10 format. The only other options valid with /TOPS are /BRIEF and /FAST.

/VOLUMEID – Use this option to display the volume identification of a particular device. The following example displays the volume ID of device DK:.

```
.DIRECTORY/VOLUMEID

 VOL ID=BL10
 OWNER NAME=JOYCE
```

The DUMP command can print on the terminal or line printer, or write to a file all or any part of a file in octal words, octal bytes, ASCII characters, and/or Radix-50 characters. It is particularly useful for examining directories and files that contain binary data.



In the command syntax shown above, filespec represents the device or file you need to examine. If you do not specify an output file, the listing prints on the line printer. If you do not specify a file type for an output file, the system uses .DMP. You can specify the entire command on one line, or you can rely on the system to prompt you for information. The DUMP command prompt is Device or file?.

Notice that some of the options (/ONLY, /START, and /END) accept a block number as an argument. Remember that all block numbers are in octal, and that the first block of a device or file is block 0. To specify a decimal block number, follow the number by a decimal point. If you are dumping a file, the block numbers you specify are relative to the beginning of that file. If you are dumping a device, the block numbers are the absolute (physical) block numbers on that device.

The system handles operations that involve magtape and cassette differently from operations involving random access devices. If you dump an RT-11 file-structured tape and specify only a device name in the file specification, the system reads only as far as the logical end-of-tape. Logical end-of-tape is indicated by an end-of-file label followed by two tape marks. For non-file-structured tape, logical end-of-tape is indicated by two consecutive tape marks. If you dump a cassette and specify only the device name in the file specification, the results are unpredictable. For magtape dumps, tape mark messages appear in the ouptut listing as the system encounters them on the tape.

The following sections describe the options you can use with the DUMP command. Following the options are some sample listings and an explanation of how to interpret them.

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

/ASCII – This option prints the ASCII equivalent of each octal word or byte that is dumped. A dot (.) represents characters that are not printable. This is the default operation.

/NOASCII – Use this option to suppress the ASCII output, which appears in the right hand column of the listing. This allows the listing to fit in 72 columns.

/BYTES – Use this option to display information in octal bytes.

/END:block – Use this option to specify an ending block number for the dump. The system dumps the device or file you specify beginning with block 0 (unless you use /START) and continuing until it dumps the block you specify with /END.

/IGNORE – Use this option to ignore errors that occur during a dump operation. Use /IGNORE if an input error occurred when you tried to perform a normal dump operation.

/ONLY:block – Use this option to dump only the block number you specify.

/OUTPUT:filespec – Use this option to specify a device and file name for the output listing file. Normally, the listing appears on the line printer. If you omit the file type for the listing file, the system uses .DMP.

/PRINTER – This option causes the output listing to appear on the line printer. This is the default operation.

/RAD50 – This option prints the Radix-50 equivalent of each octal word that is dumped.

/START:block – Use this option to specify a starting block number for the dump. The system dumps the device or file beginning at the block number you specify with /START and continuing to the end of the device or file (unless you use /END).

/TERMINAL – This option causes the output listing to appear on the console terminal. Normally, the listing appears on the line printer.

/WORDS – This option displays information in octal words. This is the default operation.

The following command dumps block 1 of the file SYSMAC.MAC. The output listing, which shows octal bytes and their ASCII equivalent, is stored in file MACLIB.DMP. The PRINT command prints the contents of the file on the line printer.

```
.DUMP/OUTPUT:MACLIB/BYTES/ONLY:1 SYSMAC.MAC

.PRINT MACLIB.DMP

DK:SYSMAC.MAC
BLOCK NUMBER 00001
000/ 040 124 117 040 124 110 105 123 105 040 114 111 103 105 116 123
       T   O       T   H   E   S   E       L   I   C   E   N   S
020/ 105 040 124 105 122 115 123 056 040 124 111 124 114 105 040 124
       E       T   E   R   M   S   .       T   I   T   L   E       T
040/ 117 040 101 116 104 040 117 127 116 105 122 123 110 111 120 040
       O       A   N   D       O   W   N   E   R   S   H   I   P
060/ 117 106 040 124 110 105 040 015 012 073 040 123 117 106 124 127
       O   F       T   H   E       .   .   :       S   O   F   T   W
100/ 101 122 105 040 123 110 101 114 114 040 101 124 040 101 114 114
       A   R   E       S   H   A   L   L       A   T       A   L   L
120/ 040 124 111 115 105 123 040 122 105 115 101 111 116 040 111 116
       T   I   M   E   S       R   E   M   A   I   N       I   N
140/ 040 104 111 107 111 124 101 114 056 015 012 073 015 012 073 040
       D   I   G   I   T   A   L   .   .   .   :   .   .   :
160/ 124 110 105 040 111 116 106 117 122 115 101 124 111 117 116 040
       T   H   E       I   N   F   O   R   M   A   T   I   O   N
200/ 111 116 040 124 110 111 123 040 123 117 106 124 127 101 122 105
       I   N       T   H   I   S       S   O   F   T   W   A   R   E
220/ 040 111 123 040 123 125 102 112 105 103 124 040 124 117 015 012
       I   S       S   U   B   J   E   C   T       T   O   .   .
240/ 073 040 103 110 101 116 107 105 040 127 111 124 110 117 125 124
       :       C   H   A   N   G   E       W   I   T   H   O   U   T
260/ 040 116 117 124 111 103 105 040 101 116 104 040 123 110 117 125
       N   O   T   I   C   E       A   N   D       S   H   O   U
```

```
300/ 114 104 040 116 117 124 040 102 105 040 103 117 116 123 124 122
     L   D       N   O   T       B   E       C   O   N   S   T   R
320/ 125 105 104 015 012 073 040 101 123 040 101 040 103 117 115 115
     U   E   D   .   .   ;       A   S       A       C   O   M   M
340/ 111 124 115 105 116 124 040 102 131 040 104 111 107 111 124 101
     I   T   M   E   N   T       B   Y       D   I   G   I   T   A
360/ 114 040 105 121 125 111 120 115 105 116 124 040 103 117 122 120
     L       E   Q   U   I   P   M   E   N   T       C   O   R   P
400/ 117 122 101 124 111 117 116 056 015 012 073 015 012 073 040 104
     O   R   A   T   I   O   N   .   .   .   ;   .   .   ;       D
420/ 111 107 111 124 101 114 040 101 123 123 125 115 105 123 040 116
     I   G   I   T   A   L       A   S   S   U   M   E   S       N
440/ 117 040 122 105 123 120 117 116 123 111 102 111 114 111 124 131
     O       R   E   S   P   O   N   S   I   B   I   L   I   T   Y
460/ 040 106 117 122 040 124 110 105 040 125 123 105 015 012 073 040
         F   O   R       T   H   E       U   S   E   .   .   ;
500/ 117 122 040 122 105 114 111 101 102 111 114 111 124 131 040 117
     O   R       R   E   L   I   A   B   I   L   I   T   Y       O
520/ 106 040 111 124 123 040 123 117 106 124 127 101 122 105 040 117
     F       I   T   S       S   O   F   T   W   A   R   E       O
540/ 116 040 105 121 125 111 120 115 105 116 124 015 012 073 040 127
     N       E   Q   U   I   P   M   E   N   T   .   .   ;       W
560/ 110 111 103 110 040 111 123 040 116 117 124 040 123 125 120 120
     H   I   C   H       I   S       N   O   T       S   U   P   P
600/ 114 111 105 104 040 102 131 040 104 111 107 111 124 101 114 056
     L   I   E   D       B   Y       D   I   G   I   T   A   L   .
620/ 015 012 073 015 012 073 040 105 106 054 112 104 054 114 120 054
     .   .   ;   .   .   ;       E   F   ,   J   D   ,   L   P   ,
640/ 102 103 054 104 126 054 103 122 054 110 112 015 012 014 056 115
     B   C   ,   D   V   ,   C   R   ,   H   J   .   .   .   .   M
660/ 101 103 122 117 040 056 056 126 061 056 056 015 012 056 115 103
     A   C   R   O       .   .   V   1   .   .   .   .   .   M   C
700/ 101 114 114 011 056 056 056 103 115 060 054 056 056 056 103 115
     A   L   L   .   .   .   .   C   M   0   ,   .   .   .   C   M
720/ 061 054 056 056 056 103 115 062 054 056 056 056 103 115 063 054
     1   ,   .   .   .   C   M   2   ,   .   .   .   C   M   3   ,
740/ 056 056 056 103 115 064 054 056 056 056 103 115 065 054 056 056
     .   .   .   C   M   4   ,   .   .   .   C   M   5   ,   .   .
760/ 056 103 115 066 015 012 056 056 056 126 061 075 061 056 015 012
     .   C   M   6   .   .   .   .   .   V   1   =   1   .   .   .
```

In the printout above, the heading shows which file was dumped and which block of the file follows. The numbers in the leftmost column indicate the byte offset from the beginning of the block. Remember that these are all octal values, and that there are two bytes per word. The octal bytes that were dumped appear in the next eight columns. The ASCII equivalent of each octal byte appears underneath the byte. The system substitutes a dot (.) for non-printing codes, such as those for control characters.

The last example shows block 6 (the directory) of device RK0:. The output is in octal words with Radix-50 equivalents below each word.

```
.DUMP/NOASCII/RAD50/ONLY:6 RK0:
```

```
RK01/N/X/016
BLOCK NUMBER  00006
000/ 000020 000002 000005 000000 000046 002000 071105 055202
       P      B      E               &      YX     RKM    NSJ
020/ 075273 000130 000015 012105 002000 071105 054162 075273
     SYS     BH      M     CI/     YX     RKM    NFB    SYS
040/ 000141 000015 012105 002000 071105 055515 075273 000150
      BQ      M     CI/     YX     RKM    NXM    SYS     BX
060/ 000015 012105 002000 015425 055202 075273 000132 000015
       M     CI/     YX    DMM    NSJ    SYS     BJ      M
100/ 012105 002000 015425 054162 075273 000143 000015 012105
     CI/     YX     DMM    NFB    SYS     BS      M     CI/
120/ 002000 015425 055515 075273 000152 000015 012105 002000
      YX    DMM    NXM    SYS     BZ      M     CI/     YX
140/ 016315 055202 075273 000130 000015 012105 002000 016315
     DXM    NSJ    SYS     BH      M     CI/     YX    DXM
160/ 054162 075273 000141 000015 012105 002000 016315 055515
     NFB    SYS     BQ      M     CI/     YX    DXM    NXM
200/ 075273 000141 000015 012105 002000 016055 055202 075273
     SYS     BQ      M     CI/     YX    DTM    NSJ    SYS
220/ 000130 000015 012105 002000 016055 054162 075273 000141
      BH      M     CI/     YX    DTM    NFB    SYS     BQ
240/ 000015 012105 002000 016055 055515 075273 000151 000015
       M     CI/     YX    DTM    NXM    SYS     BY      M
260/ 012105 002000 016005 055202 075273 000130 000015 012105
     CI/     YX     DSM    NSJ    SYS     BH      M     CI/
300/ 002000 016005 054162 075273 000141 000015 012105 002000
      YX    DSM    NFB    SYS     BQ      M     CI/     YX
320/ 016005 055515 075273 000150 000015 012105 002000 015615
     DSM    NXM    SYS     BX      M     CI/     YX    DPM
340/ 055202 075273 000130 000015 012105 002000 015615 054162
     NSJ    SYS     BH      M     CI/     YX    DPM    NFB
360/ 075273 000141 000015 012105 002000 015615 055515 075273
     SYS     BQ      M     CI/     YX    DPM    NXM    SYS
400/ 000151 000015 012105 002000 070575 055202 075273 000130
      BY      M     CI/     YX    RFM    NSJ    SYS     BH
420/ 000015 012105 002000 070575 054162 075273 000141 000015
       M     CI/     YX    RFM    NFB    SYS     BQ      M
440/ 012105 002000 070575 055515 075273 000150 000015 012105
     CI/     YX     RFM    NXM    SYS     BX      M     CI/
460/ 002000 071105 056573 075273 000123 000015 012105 002000
      YX    RKM    N&K    SYS     BC      M     CI/     YX
500/ 016315 056573 075273 000123 000015 012105 002000 016040
     DXM    N&K    SYS     BC      M     CI/     YX    DT
520/ 000000 075273 000002 000015 012105 002000 015600 000000
            SYS     B      M     CI/     YX    DP
540/ 075273 000002 000015 012105 002000 016300 000000 075273
     SYS     B      M     CI/     YX    DX             SYS
560/ 000003 000015 012105 002000 070560 000000 075273 000002
       C      M     CI/     YX    RF             SYS     B
600/ 000015 012105 002000 071070 000000 075273 000002 000015
       M     CI/     YX    RK             SYS     B      M
620/ 012105 002000 015410 000000 075273 000004 000015 012105
     CI/     YX     DM             SYS     D      M     CI/
640/ 002000 015770 000000 075273 000002 000015 012105 002000
      YX    DS             SYS     B      M     CI/     YX
```

```
660/ 100040 000000 075273 000002 000015 012105 002000 046600
     TT             SYS      B       M     CI/      YX     LP
700/ 000000 075273 000002 000015 012105 002000 012620 000000
            SYS      B       M     CI/      YX     CR
720/ 075273 000003 000015 012105 002000 052140 000000 075273
     SYS      C       M     CI/      YX     MT             SYS
740/ 000010 000015 012105 002000 051510 000000 075273 000011
       H       M     CI/      YX     MM             SYS      I
760/ 000015 012105 002000 054540 000000 075273 000002 000015
       M     CI/      YX     NL             SYS      B       M
```

The E (Examine) command prints in octal the contents of an address on the console terminal.

E (SP) address[-address]

In the command syntax illustrated above, address represents an octal address that, when added to the relocation base value from the Base command (if you used one), provides the actual address that the system examines. This command permits you to open specific locations in memory and inspect their contents. It is most frequently used after a GET command to examine locations in a program.

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

If you specify more than one address (in the form address1-address2), the system prints the contents of address1 through address2, inclusive. The second address (address2) must always be greater than the first address. If you do not specify an address, the system prints the contents of relative location 0.

Note that you cannot examine addresses outside the background area.

The following example prints the contents of location 1000, assuming the relocation base is 0.

```
.E 1000

127401
```

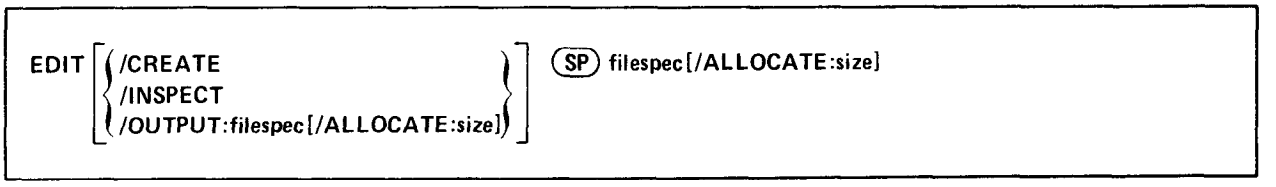
The next command sets the relocation base to 1000.

```
.B 1000
```

The following command prints the contents of locations 2000 through 2005.

```
.E 1001-1005

127401 007624 127400
```

The EDIT command invokes the text editor.



The text editor is a program that creates or modifies ASCII text files or source files for use as input to programs such as the MACRO assembler or the FORTRAN compiler. The editor reads ASCII files from any input device, makes specified changes and writes the file on any output device. It also allows efficient use of VT11 or VS60 display hardware, if this is part of the system configuration.

The editor considers a file to be divided into logical units called pages. A page of text is generally 50-60 lines long (delimited by form feed characters) and corresponds approximately to a physical page of a program listing. The editor reads one page of text at a time from the input file into its internal buffers where the page becomes available for editing. You can then use editing commands to:

- Locate text to be changed
- Execute and verify the changes
- List an edited page on the console terminal
- Output a page of text to the output file.

In the command syntax illustrated above, filespec represents the file you need to edit. You can enter the EDIT command on one line, or you can rely on the system to prompt you for information. If you do not supply a file specification for the file to edit, the system prompts you with File?. If you do not specify any option with the EDIT command, the text editor performs an edit backup operation on the file you name in the file specification. To do this, it changes the name of the original file, giving it a file type of .BAK when you finish making your editing changes. The actual file renaming occurs when you successfully exit by using an EX, EF, or EB command. You can also perform an edit backup operation while you are working with the text editor by using the Edit Backup (EB) command, which is described in Chapter 5.

When you issue an EDIT command, the system invokes the text editor. It is possible to receive an error or warning message as a result of this command. If, for example, the file you need to edit does not exist on device DK:, the editor issues an error message and remains in control.

```
.EDIT/INSPECT EXAMP3.TXT
?EDIT-F-File not found
*^C$$
```

When a situation like this occurs, you can either issue another command directly to the text editor or enter CTRL/C followed by two ESCAPEs to return control to the monitor.

The following sections describe the options you can use with the EDIT command. A more complete description of the text editor is contained in Chapter 5.

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

/CREATE – Use this option to build a new file. You can also create a new file while you are working with the text editor by using the Edit Write (EW) command, which is described in Chapter 5. The following example creates a file called NEWFIL.TXT on device DK:, inserts one line of text, and then closes the file.

```
.EDIT/CREATE NEWFIL.TXT
*ITHIS IS A NEW FILE.
$$
*EX$$
```

/INSPECT – Use this option to open a file for reading. This option does not create any new output files. You can also open a file for inspection while you are working with the text editor by using the Edit Read (ER) command, which is explained in Chapter 5.

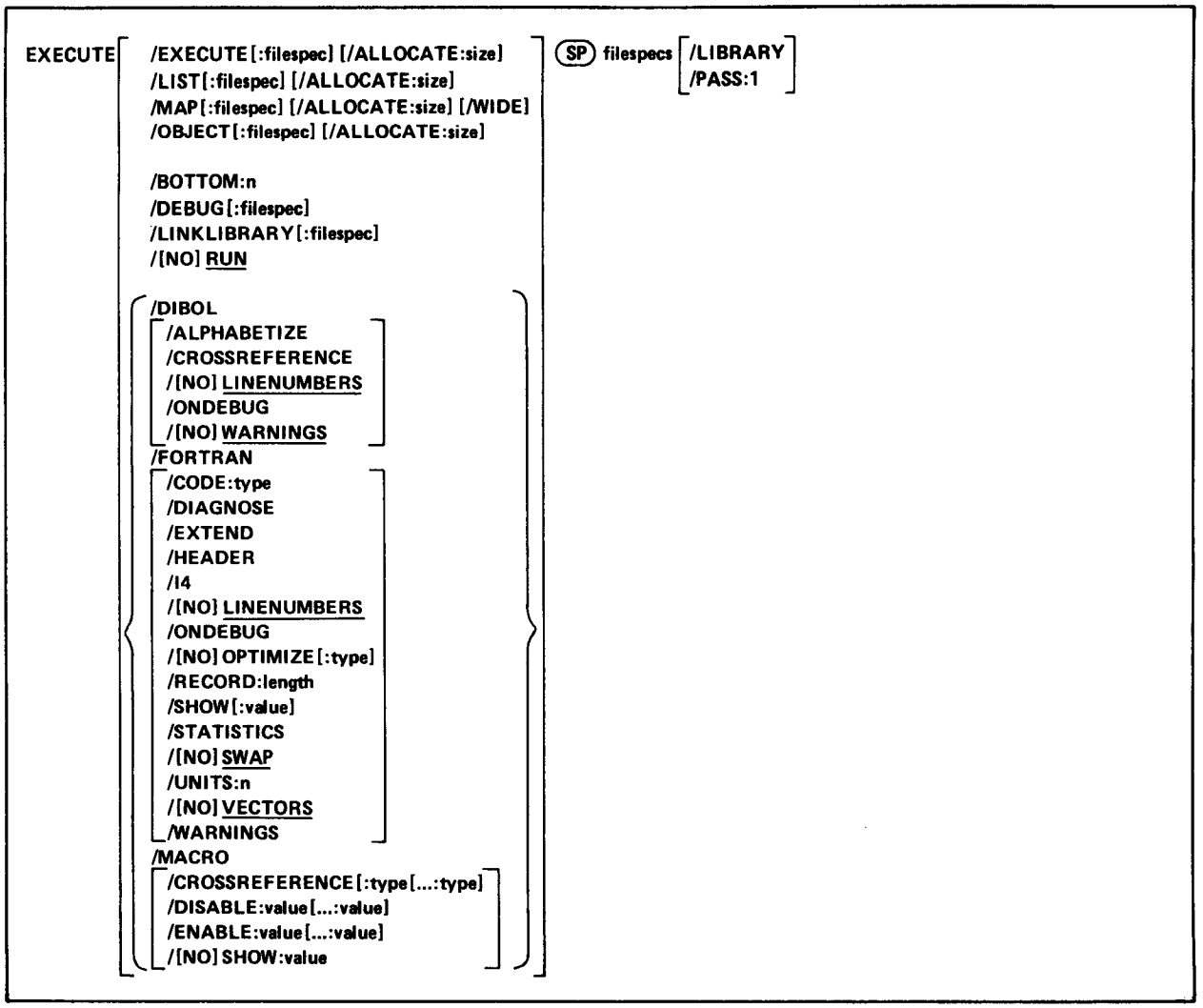
The following command opens an existing file for inspection, lists its contents, and then exits.

```
.EDIT/INSPECT NEWFIL.TXT
*R$$
*/L$$
THIS IS A NEW FILE.
*^C$$
```

/OUTPUT:filespec – This option directs the text you edit to the file you specify, leaving the input file unchanged. You can also write text to an output file while you are working with the text editor by using the Edit Write (EW) command, which is explained in Chapter 5. The following command reads file ORIG.TXT and writes the edited text to file CHANGE.TXT.

```
.EDIT/OUTPUT:CHANGE.TXT  ORIG.TXT
*
```

The EXECUTE command invokes one or more language processors to assemble or compile the files you specify. It also links object modules and initiates execution of the resultant program.



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

To compile (or assemble) multiple source files into a single object file, separate the files by plus (+) signs in the command line. Unless you specify otherwise, the system creates an object file with the same name as the first input file and gives it an .OBJ file type. To compile multiple files in independent compilations, separate the files by commas (,) in the command line. This generates a corresponding object file for each set of input files. The system then links together all the object files and creates a single executable file. You can combine up to six files for a compilation producing a single object file. You can specify the entire EXECUTE command as one line, or you can rely on the system to prompt you for information. The EXECUTE command prompt is Files?.

There are several ways to establish which language processor the EXECUTE command invokes. One way is to specify a language-name option, such as /MACRO, which invokes the MACRO assembler. Another way is to omit the language-name option and explicitly specify the file type for the source files. The EXECUTE command then invokes the language processor that corresponds to that file type. Specifying the file SOURCE.MAC, for example, invokes the MACRO assembler. A third way to establish the language processor is to let the system choose a file type of .MAC, .DBL, or .FOR for the source file you name.

To do this, the handler for the device you specify must be loaded. If you specify DX1:A, and the DX handler is loaded, the system searches for source files A.MAC and A.DBL, in that order. If it finds one of these files, the system invokes the corresponding language processor. If it cannot find one of these files, or if the device handler associated with the input file is not resident, the system assumes a file type of .FOR and invokes the FORTRAN compiler.

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

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

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

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

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

/BOTTOM:n – Use this option to specify the lowest address to be used by the relocatable code in the load module. The argument, n, represents a 6-digit unsigned even octal number. If you do not use this option, the system positions the load module so that the lowest address is location 1000 (octal). This option is illegal for foreground links.

/CODE:type – Use this option with FORTRAN to produce object code that is designed for a particular hardware configuration. The argument, type, represents a three-letter abbreviation for the type of code to produce. The legal values are the following: EAE, EIS, FIS, and THR. See Section 1.1.1, Compiler Generated Code, of the *RT-11/RSTS/E FORTRAN IV User's Guide* for a complete description of the types of code and their function.

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

With MACRO, this option takes an optional argument. The argument, type, represents a one-character code that indicates which sections of the cross-reference listing the assembler should include. Table 4-10 summarizes the valid arguments and their meaning.

/DEBUG[:filespec] – Use this option to link ODT (online debugging technique, described in Chapter 16) with your program to help you debug it. If you supply the name of another debugging program, the system links the debugger you specify with your program. The debugger is always linked low in memory relative to your program.

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

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

/DISABLE:value[. . . :value] – Use this option with MACRO to specify a .DSABL directive. Table 4-11 summarizes the arguments and their meaning. See Section 6.2 of the *PDP-11 MACRO Language Reference Manual* for a description of the directive and a list of all legal values.

/ENABLE:value[. . . :value] – Use this option with MACRO to specify an .ENABL directive. Table 4-11 summarizes the arguments and their meaning. See Section 6.2 of the *PDP-11 MACRO Language Reference Manual* for a description of the directive and a list of all legal values.

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

```
.EXECUTE  MYPROG

.EXECUTE/EXECUTE  MYPROG
```

Both commands link MYPROG.OBJ and produce MYPROG.SAV as a result. The /EXECUTE option has different meanings when it follows the command and when it follows the file specification. The following command creates an executable file called PROG1.SAV on device RK1:.

```
.EXECUTE/EXECUTE:RK1:  PROG1,PROG2
```

The next command creates an executable file called MYPROG.SAV on device DK:.

```
.EXECUTE  RTN1,RTN2,MYPROG/EXECUTE
```

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

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

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

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

/LIBRARY – Use this option with MACRO to identify the file the option qualifies as a macro library file. Use it only after a library file specification in the command line. The MACRO assembler looks first to the library associated with the most recent /LIBRARY option to satisfy references (made with the .MCALL directive) from MACRO programs. It then looks to any libraries you specified earlier in the command line, and it looks last to SYSMAC.SML.

In the example below, the two files A.FOR and B.FOR are compiled together, producing B.OBJ and B.LST. The MACRO assembler assembles C.MAC, satisfying .MCALL references from MYLIB.MAC and SYSMAC.SML. It produces C.OBJ and C.LST. The system then links B.OBJ and C.OBJ together, resolving undefined references from SYSLIB.OBJ and produces the executable file B.SAV. Finally, the system loads and executes B.SAV.

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

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

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

/LINKLIBRARY:filespec – Use this option to include the library file name you specify as an object module library during the linking operation. Repeat the option if you need to specify more than one library file.

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

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

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

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

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

If the /LIST option contains a name and file type to override the default of .LST, the system generates a listing file with that name. The following command, for example, compiles A.FOR and B.FOR together, producing files A.OBJ and FILE1.OUT on device DK:. It then links A.OBJ (using SYSLIB.OBJ as needed) and produces A.SAV.

```
.EXECUTE/NORUN/FORTRAN/LIST:FILE1.OUT A+B
```

You cannot use a command line like the next one. In this example, the second listing file would replace the first one and, therefore, cause an error.

```
.EXECUTE/LIST:FILE2 A.MAC,B.MAC
```

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

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

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

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

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

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

```
.EXECUTE/NORUN A.MAC/LIST,B.FOR
```

This command compiles A.MAC, producing A.OBJ and A.LST. It also compiles B.FOR, producing B.OBJ. However, it does not produce any listing file for the compilation of B.FOR. Finally, the system links A.OBJ and B.OBJ together, producing A.SAV.

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

/MAP[:filespec] – You must specify this option to produce a load map after a link operation. The /MAP option has different meanings depending on where you put it in the command line. It follows the same general rules outlined above for /LIST.

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

```
.EXECUTE/FORTRAN A

.EXECUTE/FORTRAN/OBJECT A
```

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

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

```
.EXECUTE/OBJECT:RK1: A.MAC,B.MAC
```

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

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

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

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

/OPTIMIZE:type – Use this option with FORTRAN to enable certain options that optimize object code for various conditions. The value, type, represents the three-letter code for the type of optimization to enable. Table 4-4 summarizes the codes and their meanings.

/NOOPTIMIZE:type – Use this option with FORTRAN to disable certain options that optimize object code for various conditions. The value, type, represents the three-letter code for the type of optimization to disable. Table 4-4 summarizes the codes and their meanings.

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

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

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

/RUN – Use this option to initiate execution of your program if there are no errors in the compilation or the link. This is the default operation.

/NORUN – Use this option to suppress execution of your program. The system performs only the compilation and the link.

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

Use this option with MACRO to specify any MACRO .LIST directive. Table 4-12 summarizes the valid arguments and their meaning. Section 6.1.1, .LIST and .NLIST Directives, of the *PDP-11 MACRO Language Reference Manual* explains how to use these directives.

/NOSHOW:value – Use this option with MACRO to specify any MACRO .NLIST directive. Table 4-12 summarizes the valid arguments and their meaning. Section 6.1.1, .LIST and .NLIST Directives, of the *PDP-11 MACRO Language Reference Manual* explains how to use these directives.

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

/SWAP – Use this option with FORTRAN to permit the USR (user service routine) to swap over the FORTRAN program in memory. This is the default operation.

/NOSWAP – Use this option with FORTRAN to keep the USR resident during execution of a FORTRAN program. This may be necessary if the FORTRAN program uses some of the RT-11 system subroutine library calls (see Chapter 4 of the *RT-11 Advanced Programmer's Guide*). If the program frequently updates or creates a large number of different files, making the USR resident can improve program execution. However, the penalty for making the USR resident is 2K words of memory.

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

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

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

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

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

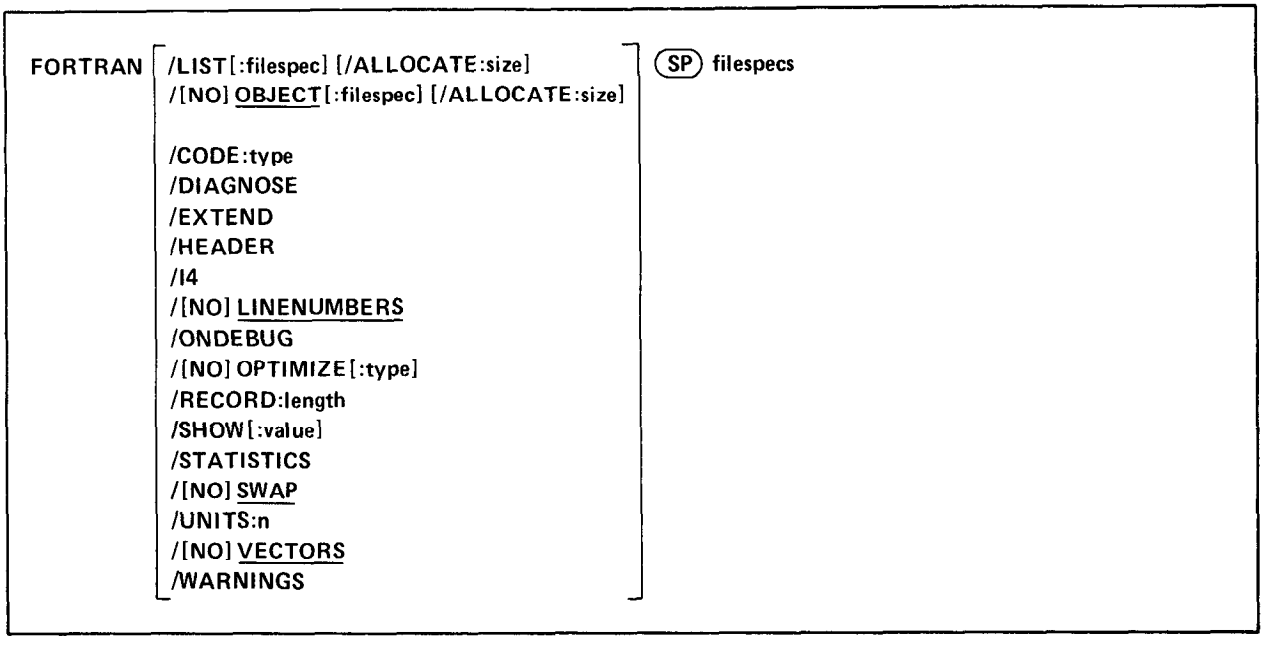
/WIDE – Use this option with /MAP to produce a wide load map listing. Normally, the listing is wide enough for three GLOBAL VALUE columns, which is suitable for paper with 72 or 80 columns. The /WIDE option produces a listing that is six GLOBAL VALUE columns wide, which is ideal for a 132-column page.

The FOCAL command invokes the FOCAL language interpreter.

FOCAL

FOCAL has its own command language. Therefore, the FOCAL command accepts no options and no file specifications.

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



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

Language options are position dependent. That is, they have different meanings depending on where you place them in the command line. Options that qualify a command name apply across the entire command string. Options that follow a file specification apply only to the file (or group of files separated by plus signs) that they follow in the command string. You can enter the FORTRAN command as one line, or you can rely on the system to prompt you for information. The FORTRAN command prompt is Files? for the input specification.

The *RT-11/RSTS/E FORTRAN IV User's Guide* contains more detailed information about using FORTRAN. The following sections describe the options you can use with the FORTRAN command.

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

/CODE:type – Use this option to produce object code that is designed for a particular hardware configuration. The argument, type, represents a three-letter abbreviation for the type of code to produce. The legal values are the following: EAE, EIS, FIS, and THR. See Section 1.1.1 of the *RT-11/RSTS/E FORTRAN IV User's Guide* for a complete description of the types of code and their functions.

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

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

/HEADER – This option includes in the printout a list of options that are currently in effect.

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

/LINENUMBERS – Use this option to include internal sequence numbers in the executable program. These are especially useful in debugging a FORTRAN program. They identify the FORTRAN statements that cause run-time diagnostic error messages. This is the default operation.

/NOLINENUMBERS – This option suppresses the generation of internal sequence numbers in the executable program. This produces a smaller program and optimizes execution speed. Use this option to compile only those programs that are already debugged; otherwise the line numbers in FORTRAN error messages are replaced by question marks and the messages are difficult to interpret.

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

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

```
.FORTRAN/LIST:TT: A
```

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

```
.FORTRAN/LIST:RK3: A
```

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

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

You cannot use a command line like the next one. In this example, the second listing file would replace the first one and, therefore, cause an error.

```
.FORTRAN/LIST:FILE2 A,B
```

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

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

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

```
.FORTRAN A/LIST:B
```

```
.FORTRAN/LIST:B A
```

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

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

```
.FORTRAN A/LIST,B
```

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

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

```
.FORTRAN A

.FORTRAN/OBJECT A
```

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

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

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

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

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

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

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

/ONDEBUG – Use this option to include debug lines (those that have a D in column one) in the compilation. You do not, therefore, have to edit the file to include these lines in the compilation or to logically remove them. This option is useful in debugging a program. You can include messages, flags, and conditional branches to help you trace program execution and find an error.

/OPTIMIZE:type – Use this option to enable certain options that optimize object code for various conditions. The argument, type, represents the three-letter code for the type of optimization to enable. Table 4-4 summarizes the codes and their meanings.

Table 4-4 Optimization Codes

Code	Meaning
BND	Global register bindings for inline code generation
CSE	Common subexpression elimination
SPD	Optimization for speed of execution as opposed to minimal program size
STR	Strength reduction optimization

/NOOPTIMIZE:type – Use this option to disable certain options that optimize object code for various conditions. The argument, type, represents the three-letter code for the type of optimization to disable. Table 4-4 summarizes the codes and their meanings.

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

/SHOW[:value] – Use this option to control FORTRAN listing output. The argument, value, represents a code that indicates which listings the compiler is to produce. Table 4-5 summarizes the codes and their meaning. You can combine options by specifying the sum of their numeric codes. For example:

```
/SHOW:7
```

or

```
/SHOW:ALL
```

The two options shown above have the same meaning. If you specify no code, the default value is 3, a combination of SRC and MAP.

Table 4-5 FORTRAN Listing Codes

Code	Meaning
0	Lists diagnostics only
1 or SRC	Lists source program and diagnostics
2 or MAP	Lists storage map and diagnostics
3	Lists diagnostics, source program, and storage map
4 or COD	Lists generated code and diagnostics
7 or ALL	Lists diagnostics, source program, storage map, and generated code

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

/SWAP – Use this option to permit the USR (user service routine) to swap over the FORTRAN program in memory. This is the default operation.

/NOSWAP – This option keeps the USR resident during execution of a FORTRAN program. This may be necessary if the FORTRAN program uses some of the RT-11 System Subroutine Library calls (see Chapter 4 of the *RT-11 Advanced Programmer's Guide*). If the program frequently updates or creates a large number of different files, making the USR resident can improve program execution. However, the penalty for making the USR resident is 2K words of memory.

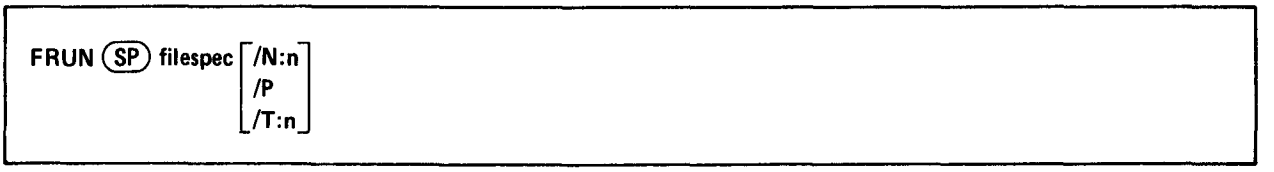
/UNITS:n – Use this option to override the default number of logical units (6) to be open at one time. The maximum value you can specify for n is 16.

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

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

/WARNINGS – Use this option to include warning messages in FORTRAN compiler diagnostic error messages. These messages call certain conditions to your attention, but do not interfere with the compilation. A warning message prints, for example, if you change an index within a DO loop, or if you specify a variable name longer than six characters.

The FRUN command initiates foreground jobs.



In the command syntax illustrated above, filespec represents the program to execute. Because this command runs a foreground job, it is valid for the FB and XM monitors only.

If another foreground job is active when you issue the FRUN command, an error message prints on the terminal. You can run only one foreground job at a time. If a terminated foreground job is occupying memory, the system reclaims that region for your program. Then, if the system finds your program and if your program fits in the available memory, execution begins.

The following sections describe the options you can use with FRUN. Note that the option must follow the file specification in the command line.

/N:n – Use this option to reserve space in memory over the actual program size. The argument, n, represents the number of words of memory to allocate. You must use this option to execute a FORTRAN foreground job.

/P – Use this option to help you debug a program. When you type the carriage return at the end of the command string, the system prints the load address of your program and waits. You can examine or modify the program (by using ODT, described in Chapter 16) before starting execution. You must use the RESUME command to restart the foreground job. The following command loads the program DEMOSP.REL, prints the load address, and waits for a RESUME command to begin execution.

```
.FRUN DEMOSP/P
Loaded at 127276
.RESUME
```

/T:n – Use this option to assign a terminal to interact with the foreground job. The argument, n, represents a terminal logical unit number. The default value is 0, which represents the original console terminal. By assigning a different terminal to interact with the foreground job, you eliminate the need for the foreground and background jobs to share the console terminal. Note that the original console terminal still interacts with the background job and with the keyboard monitor, unless you use the SET TT: CONSOL command to change this.

The GET command loads a memory image file into memory.

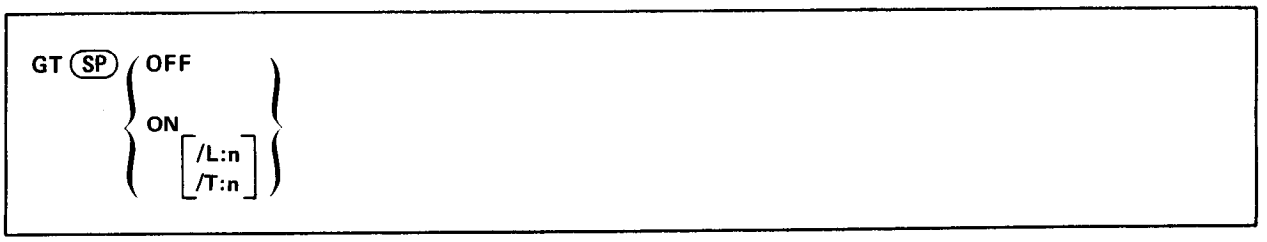
GET (SP) filespec

In the command syntax shown above, filespec represents the memory image file to be loaded. The default file type is .SAV. Note that magtape and cassette are not block-replaceable devices and, therefore, are not permitted with the GET command. Use the GET command for a background job only. You cannot use GET on a virtual program that executes under the XM monitor. The GET command is useful when you need to modify or debug a program. You can use GET with the Base, Deposit, Examine, and START commands to test changes. Use the SAVE command to make these changes permanent. You can combine programs by issuing multiple GET commands, as the following example shows. This example loads a program, DEMOSP.SAV, loads ODT.SAV (on-line debugging technique, described in Chapter 16), and starts the program using the address of ODT's entry point, O.ODT.

```
.GET DEMOSP

.GET ODT

.START

ODT V01.04
*
```

If more than one program requires the same locations in memory, the program you load later overlays the previous program. Note that you cannot use GET to load overlay segments of a program; it can load only the root. If the file you need to GET resides on a device other than the system device, the system automatically loads that device handler into memory when you issue the GET command. This prevents problems from occurring if you use the START command and your program is overlaid.

The GT command enables or disables the VT11 or VS60 display hardware.



When you issue the GT OFF command, you disable the display hardware. The printing console terminal then becomes the device that transmits your commands to the system.

When you issue the GT ON command, the display screen replaces the printing console terminal. The display screen offers some advantages over the printing terminal: 1) it is quieter than a printing terminal, 2) it is faster than a printing terminal, 3) it does not require a supply of paper, and 4) it is the device for which the text editor's immediate mode is intended. The display screen can speed up the editing process (see Chapter 5 for information on how to use the text editor). You can use CTRL/A, CTRL/S, CTRL/E, and CTRL/Q to control scrolling. These commands are explained in Section 3.6. Note that RT-11 does not permit you to use display hardware (with GT ON) in an 8K configuration. You cannot issue GT ON when a foreground job is active; this causes the system to print an error message. Issue the GT ON command before you begin execution of the foreground job. ODT (on-line debugging technique, described in Chapter 16) is the only system program that cannot use the display screen. Its output always appears on the console terminal.

Table 4-6 Display Screen Values

Screen Size	Lines	Top Position
12 inch	1-31	1-744
17 inch (or larger)	1-40	1-1000

The following options let you control the number of lines that appear on the screen and position the first line vertically.

/L:n – Use this option to change the number of lines of text that display on the screen. Table 4-6 shows the valid range for the argument, n, in decimal. If you do not use this option, the system determines the screen size and automatically assigns the largest valid value.

/T:n – Use this option to change the top position of the scroll display. Table 4-6 shows the valid range for the argument, n, in decimal. If you do not use this option, the system determines the screen size and automatically assigns the largest valid value.

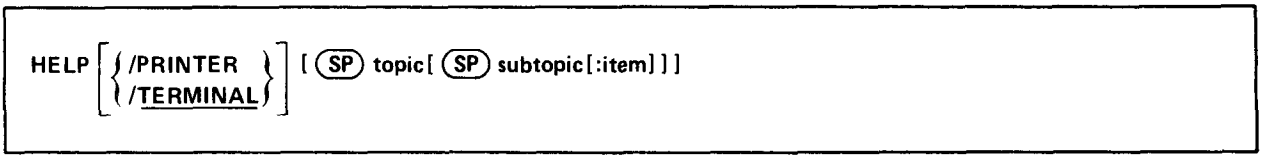
The following command enables the display screen.

```
.GT ON
```

The next command disables the display screen.

```
.GT OFF
```

The HELP command lists useful information.



In the command syntax shown above, topic represents a specific subject about which you need information. In the help file supplied with RT-11, the topics are the keyboard monitor commands. The subtopic represents a specific category within a topic. In the RT-11 help file, the subtopics are syntax, semantics, options, and examples. The item represents one member of the subtopic group. You can specify more than one item in the command line if you separate the items by colons (:).

The HELP command permits you to access the file HELP.TXT. The help file distributed with RT-11 contains information about the keyboard monitor commands and how to use them. However, the concept of the help file is a general one. That is, you can create your own help file to supply quick reference material on any subject. Structure your HELP.TXT file in the same format as the standard RT-11 HELP.TXT. Note that the HELP command reads the file that is specifically named HELP.TXT. There are only two options you can use with the HELP command. They are /PRINTER and /TERMINAL.

/PRINTER – Use this option to list helpful information on the line printer.

/TERMINAL – This option lists helpful information on the console terminal. This is the default operation.

The following examples all make use of the standard RT-11 help file.

The following command lists all the topics for which assistance is available.

```
.HELP *

HELP        Lists helpful information
APL         Invokes the APL language interpreter
ASSIGN      Associates a logical device name with a physical device
BASIC       Invokes the BASIC language interpreter
.
.
.
```

The next command lists all the information about the DATE command.

```
.HELP DATE

DATE        Sets or displays the current system date

  SYNTAX
            DATE[ dd-mmm-yy]

  SEMANTICS
            All numeric values are decimal;  mmm  represents  the  first
            three characters of the name of the month.
```

```
OPTIONS
        None

EXAMPLES
        DATE 12-MAY-77
```

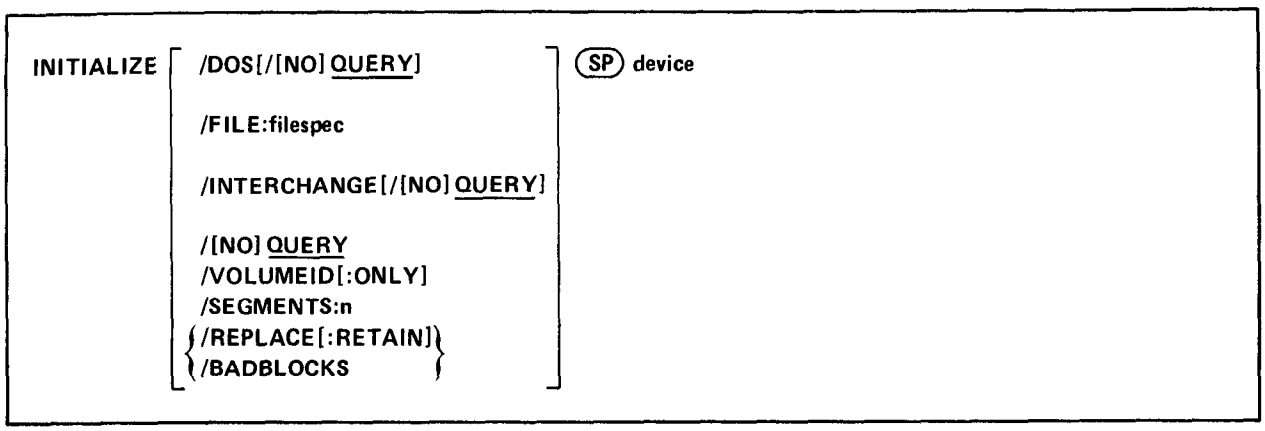
The next command lists all the options that are valid with the DIRECTORY command.

```
.HELP DIRECTORY OPTIONS

OPTIONS
  ALLOCATE:size
       Use with /OUTPUT to reserve space for the output listing file
ALPHABETIZE
       Sorts the directory in alphabetical order by file name and
       type
.
.
.
```

The last command lists information about the /BRIEF option for the DIRECTORY command.

```
.HELP DIRECTORY OPTIONS:BRIEF

  BRIEF
       Lists only file names and file types of files;  same as /FAST
```

Use the INITIALIZE command to clear and initialize a device directory.



In the command syntax illustrated above, device represents the device you need to initialize. The initialize operation must always be the first operation you perform on a new device after you receive it from the manufacturer. This procedure destroys any data that may already exist on a device. After you use the INITIALIZE command, there are no files in the directory. If you use the INITIALIZE command with no options, the system simply initializes the device directory. You can enter the INITIALIZE command as one line, or you can rely on the system to prompt you for the name of the device with Device?. The following sections describe the options you can use with INITIALIZE and give some examples of their use.

/BADBLOCKS – Use this option to scan a device (disk or DECtape) for bad blocks and write .BAD files over them. For each bad block the system encounters on the device, it creates a file called FILE.BAD to cover it. After the device is initialized and the scan completed, the directory consists only of FILE.BAD entries that cover the bad blocks. This procedure ensures that the system will not attempt to access these bad blocks during routine operations. If the system finds a bad block in either the boot block or the device directory, it prints an error message and the device is not usable. The following command initializes device RK1: and scans for bad blocks.

```
.INITIALIZE/BADBLOCKS RK1:

RK1:/Init are you sure?Y
```

/DOS – Use this option to initialize a DECtape for DOS-11 format.

/FILE:filespec – Use this option to initialize a magtape and create a bootable tape. For filespec, substitute dev:MBOOT.BOT. This file is distributed with RT-11 for this purpose only. Consult the *RT-11 System Generation Manual* for more information. The following example creates a bootable magtape:

```
.INITIALIZE/FILE:MBOOT.BOT  MT0:
```

/INTERCHANGE – Use this option to initialize a diskette for interchange (proposed ANSI standard) format. The following example initializes DX1: in interchange format.

```
.INITIALIZE/INTERCHANGE DX1:
DX1:/Z ARE YOU SURE?  Y
```

/QUERY – This option prompts you for confirmation before it initializes a device. Respond by typing a Y followed by a carriage return to initiate execution of the command. The system interprets a response beginning with any other character to mean NO. /QUERY is the default operation.

/NOQUERY – Use this option to suppress the confirmation message that the system prints before it proceeds with the initialization.

/REPLACE[:RETAIN] – Use the /REPLACE option to scan the disk for bad blocks when you initialize an RK06. If the system finds any bad blocks, it creates a replacement table so that routine operations access good blocks instead of bad ones. Thus, the disk appears to consist of only good blocks. Note, though, that accessing this replacement table slows response time for routine input and output transactions. If you use :RETAIN with /REPLACE, the system initializes the RK06 but does not create a replacement table for bad blocks. Instead, it uses the replacement table that is already on the device as a result of a previous initialization. This procedure allows the initialization to proceed faster.

/SEGMENTS:n – Use this option if you need to initialize a disk and change the number of directory segments. The number of segments in the directory determines the number of files that can be sorted on a device. The system allows a maximum of 72 files per directory segment, and 31 directory segments per device. The argument, n, represents the number of directory segments you need to create. The valid range for n is from 1 to 31 (decimal). Table 4-7 shows the default values of n for standard RT-11 devices.

Table 4-7 Default Directory Sizes

Device	Size (decimal) of Directory in Segments
RK05	16
DT	4
RF	4
DS	4
DP	31
DX	4
RK06	31

/VOLUMEID[:ONLY] – Use this option to write a volume identification on a device when you initialize it. This identification consists of a volume ID (up to 12 characters long for a block-replaceable device, up to 6 characters long for magtape) and an owner name (up to 12 characters long for a block-replaceable device, up to 10 characters long for magtape). The following example initializes device RK1: and writes a volume identification on it.

```
.INITIALIZE/VOLUMEID RK1:

RK1:/Init are you sure?Y

 VOL ID?BACKUP2

 OWNER NAME?ENGINEERING
```

Use /VOLUMEID:ONLY to write a new volume identification on a device without reinitializing the device.

The INSTALL command installs the device you specify into the system.

```
INSTALL (SP) device[ , . . . device]
```

In the command syntax shown above, device represents the name of the device to be installed. The INSTALL command accepts no options. The INSTALL command allows you to install into the system tables a device that was not originally built into the system. (A device handler must exist in the system tables before you can use that device.) The device occupies the first available device slot. Using the INSTALL command does not change the monitor disk image; it only modifies the system tables of the monitor that is currently in memory.

You can enter the command on one line, or you can rely on the system to prompt you for information. The INSTALL command prompt is Device?.

When you specify a device name, the system searches the system device for the corresponding device handler file. For SJ and FB systems, if LP: is to be installed, the INSTALL command searches for the file SY:LP.SYS. For XM systems, INSTALL searches for SY:LPX.SYS. The INSTALL command does not allow a device handler built for a different configuration of the system to be installed in a given system. For example, you cannot install an error logging handler if your currently running monitor is not designed for error logging. Note that you cannot install the following device names: FG (with FB or XM monitor only), and BA.

To permanently install a device, include the INSTALL command in the standard system startup indirect command file. This file is invoked as an indirect file automatically when you boot the system. The INSTALL command also allows you to configure a special system for a single session without having to reconfigure to get back to the standard device configuration. Rebooting the system restores the original device configuration. Note that if there are no free device slots (use the SHOW DEVICES command to determine this), you must remove an existing device (with the REMOVE command) before you can install a new device.

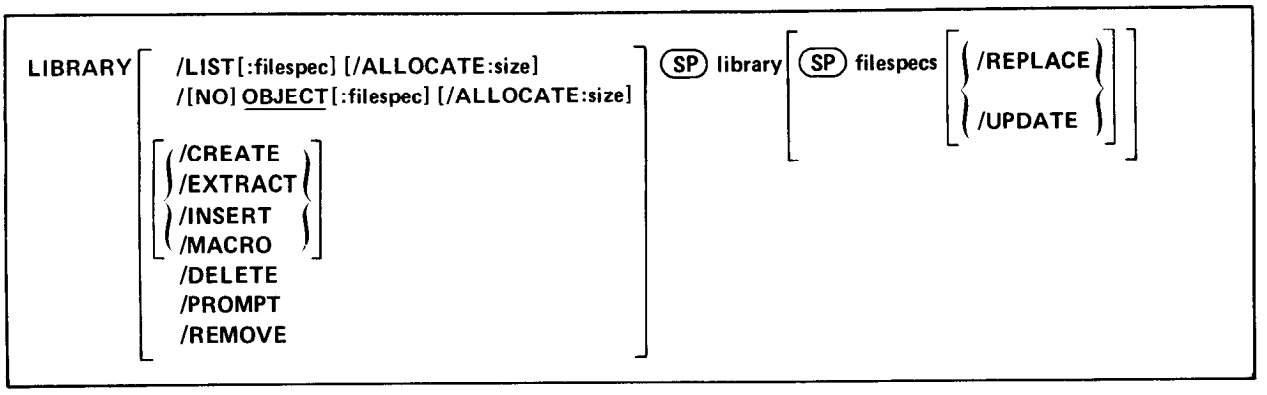
The following command installs the card reader into the system tables from the file CR.SYS. Note that the colon (:) that follows the device handler name is optional.

```
.INSTALL CR:
```

The next example installs the line printer, the card reader, and DECtape.

```
.INSTALL LP:,CR:,DT:
```

The LIBRARY command lets you create, update, modify, list, and maintain library files.



In the command syntax illustrated above, library represents the library file name and filespecs represents the input module file names. Separate the library file specification from the module file specifications by a space. Separate the module file specifications by commas. The system uses .LST as the default file type for library directory listing files. It also uses .OBJ as the default file type for object libraries and object input files, and it uses .MAC for macro libraries and macro input files. The default operation, if you do not specify an option, is /INSERT. If you do not specify a library file in the command line, the system prompts you with Library?. If you specify /CREATE, /INSERT, or /MACRO and omit the module file specification, the system prompts you with Files?. If you specify /EXTRACT, the system prompts you with File?. Note that no other options cause the File? or Files? prompts.

The LIBRARY command can perform all the functions listed above on object library files. It can also create macro library files for use with the MACRO-11 assembler. A library file is a direct access file (a file that has a directory) that contains one or more modules of the same module type. The system organizes the library files so that the linker and MACRO-11 assembler can access them rapidly. Each object library is a file that contains a library header, library directory, and one or more object modules. The object modules in a library file can be routines that are repeatedly used in a program, routines that are used by more than one program, or routines that are related and simply gathered together for convenience. The contents of the library file are determined by your needs. An example of a typical object library file is the default system library, SYSLIB.OBJ, used by the linker. An example of a macro library file is SYSMAC.SML.

You access object modules in a library file from another program by making calls or references to their global symbols; you link the object modules with the program that uses them by using the LINK command to produce a single executable module. Each input file for an object library consists of one or more object modules, and is stored on a device under a specific file name and file type. Once you insert an object module into a library file, you no longer reference the module by the file name of which it was a part. Reference it now by its individual module name. For example, the input file FORT.OBJ may exist on DT2: and can contain an object module called ABC. Once you insert the module into a library, reference only ABC and not FORT.OBJ.

The input files normally do not contain main programs but only subprograms, functions and subroutines. The library files must never contain a FORTRAN "BLOCK DATA" subprogram: there is no undefined global symbol to cause the linker to load it automatically.

The following sections describe the LIBRARY command options and explain how to use them. The last section under this command describes the LIBRARY prompting sequence and order of execution for commands that combine two or more LIBRARY options. Chapter 12 contains more detailed information on object and macro libraries. The following sections describe the options available with the LIBRARY command.

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

/CREATE – Use this option to create an object library. Specify a library name followed by the file specifications for the modules that are to be included in that library. The following command, for example, creates a library called NEWLIB.OBJ from the modules contained in files FIRST.OBJ and SECOND.OBJ.

```
.LIBRARY/CREATE NEWLIB FIRST,SECOND
```

/DELETE – Use this option to delete an object module and all its associated global symbols from the library. Specify the library name in the command line. The system prompts you for the names of the modules to delete. The prompt is:

```
Module name?
```

Respond with the name of a module. (Be sure to specify a module name and not a global name.) Follow each module name with a carriage return. Enter a carriage return on a line by itself to terminate the list of module names. The following example deletes modules SGN and TAN from the library called NEWLIB.OBJ.

```
.LIBRARY/DELETE NEWLIB
Module name? SGN
Module name? TAN
Module name?
```

/EXTRACT – Use this option to extract an object module from a library and store it in a file with the same name as the module and a file type of .OBJ. You cannot combine this option with any other option. The system prompts you for the name of the object module to be extracted. The prompt is:

```
Global?
```

If you specify a global name, the system extracts the entire module of which that global is a part. Follow each global name with a carriage return. Enter a carriage return on a line by itself to terminate the list of global symbols. The following example, which also shows the system prompts, extracts the module ATAN from the library called NEWLIB.OBJ, storing it in file ATAN.OBJ on DX1:.

```
.LIBRARY/EXTRACT
Library? NEWLIB
File    ? DX1:ATAN
Global ? ATAN
Global ?
```

/INSERT – Use this option to insert an object module into an existing library. Although you can insert two or more object modules having the same name, this practice is not recommended because of the difficulty involved in replacing or updating these modules. Note that /INSERT is the default operation. If you do not specify any option, insertion takes place. The following example inserts the modules contained in the files THIRD.OBJ and FOURTH.OBJ into the library called OLDLIB.OBJ.

```
.LIBRARY/INSERT OLDLIB THIRD,FOURTH
```

/LIST[:filespec] – Use this option to obtain a directory listing of an object library. The following example obtains a directory listing of OLDLIB.OBJ on the terminal (the line printer is the default device).

```
.LIBRARY/LIST:TT:  OLDLIB
```

The directory listing prints global symbol names. A plus sign (+) in the module column indicates a continued line. See Section 12.2.7 for a procedure to include module names in the directory listing.

You can also use /LIST with other options (except /MACRO) to obtain a directory listing of an object library after you create or modify it. The following command, for example, inserts the modules contained in the files THIRD.OBJ and FOURTH.OBJ into the library called OLDLIB.OBJ, and prints a directory listing of the library on the terminal.

```
.LIBRARY/INSERT/LIST:TT:  OLDLIB THIRD,FOURTH
```

You cannot obtain a directory listing of a macro library (see /MACRO).

/MACRO – Use this option to create a macro library. Note that this is the only valid function for a macro library. You can create a macro library, but you cannot list or modify it. To update a macro library, simply edit the ASCII text file and then reprocess the file with the LIBRARY/MACRO command. The following example creates a macro library called NEWLIB.MAC from the ASCII input file SYSMAC.MAC.

```
.LIBRARY/MACRO NEWLIB SYSMAC
```

/OBJECT[:filespec] – The system creates object library files by default as a result of executing a LIBRARY command. When you modify an existing library, the system actually makes the changes to the library you specify, thus creating a new, updated library that it stores under the same name as the original library. Use this option to give a new name to the updated library file and preserve the original library. The following example creates a library called NEWLIB.OBJ, which consists of the library OLDLIB.OBJ plus the modules that are contained in files THIRD.OBJ and FOURTH.OBJ.

```
.LIBRARY/INSERT/OBJECT:NEWLIB OLDLIB THIRD,FOURTH
```

/NOOBJECT – Use this option to suppress the creation of a new object library as a result of a LIBRARY command.

/PROMPT – Use this option to specify more than one line of input file specifications in a LIBRARY command. This option is valid with all other library functions except the /EXTRACT option. You must specify // as the last input in order to properly terminate the input list. The following example creates a macro library called MACLIB.MAC from seven input files.

```
.LIBRARY/MACRO/PROMPT MACLIB A,B,C,D
*E,F,G
*//
```

/REMOVE – This option permits you to delete a specific global symbol from a library file's directory. Since globals are only deleted from the directory (and not from the object module itself), all the globals that were previously deleted are restored whenever you update that library, unless you use /REMOVE again to delete them. This feature lets you recover a library if you have inadvertently deleted the wrong global. The system prompts you for the names of the global symbols to remove. The prompt is:

```
Global?
```

Respond with the name of a global symbol to be removed. Follow each global symbol with a carriage return. Enter a carriage return on a line by itself to terminate the list of global symbols. The following example deletes the globals GA, GB, GC, and GD from the library OLDLIB.OBJ.

```
.LIBRARY/REMOVE OLDLIB
Global? GA
Global? GB
Global? GC
Global? GD
Global?
```

/REPLACE – Use this option to replace modules in an existing object library with modules of the same name contained in the files you specify. If an old module does not exist with the same name as the input module you specify, or if you specify /REPLACE with a library file name, the system prints an error message and ignores the command. The following example replaces a module called SQRT in the library MATHLB.OBJ with a new module, also called SQRT, from the file called MFUNCT.OBJ.

```
.LIBRARY MATHLB MFUNCT/REPLACE
```

Note that the /REPLACE option must follow each file specification that contains a module to be inserted into the library.

/UPDATE – This option combines the functions of /INSERT and /REPLACE. Specify it after each file specification to which it applies. If the modules in the input file already exist in the library, the system replaces those library modules. If the modules in the input file do not exist in the library, the system inserts them. The following example updates the library OLDLIB.OBJ.

```
.LIBRARY OLDLIB FIRST/UPDATE,SECOND/UPDATE
```

You can combine the LIBRARY options with the exceptions of /EXTRACT and /MACRO, which you cannot combine with most of the other functions. Table 4-8 lists the sequence in which the system executes the LIBRARY options and prompts you for additional information.

Table 4-8 LIBRARY Execution and Prompting Sequence

Option	Prompt
/CREATE	
/DELETE	Module name?
/REMOVE	Global?
/UPDATE	
/REPLACE	
/INSERT	
/LIST	

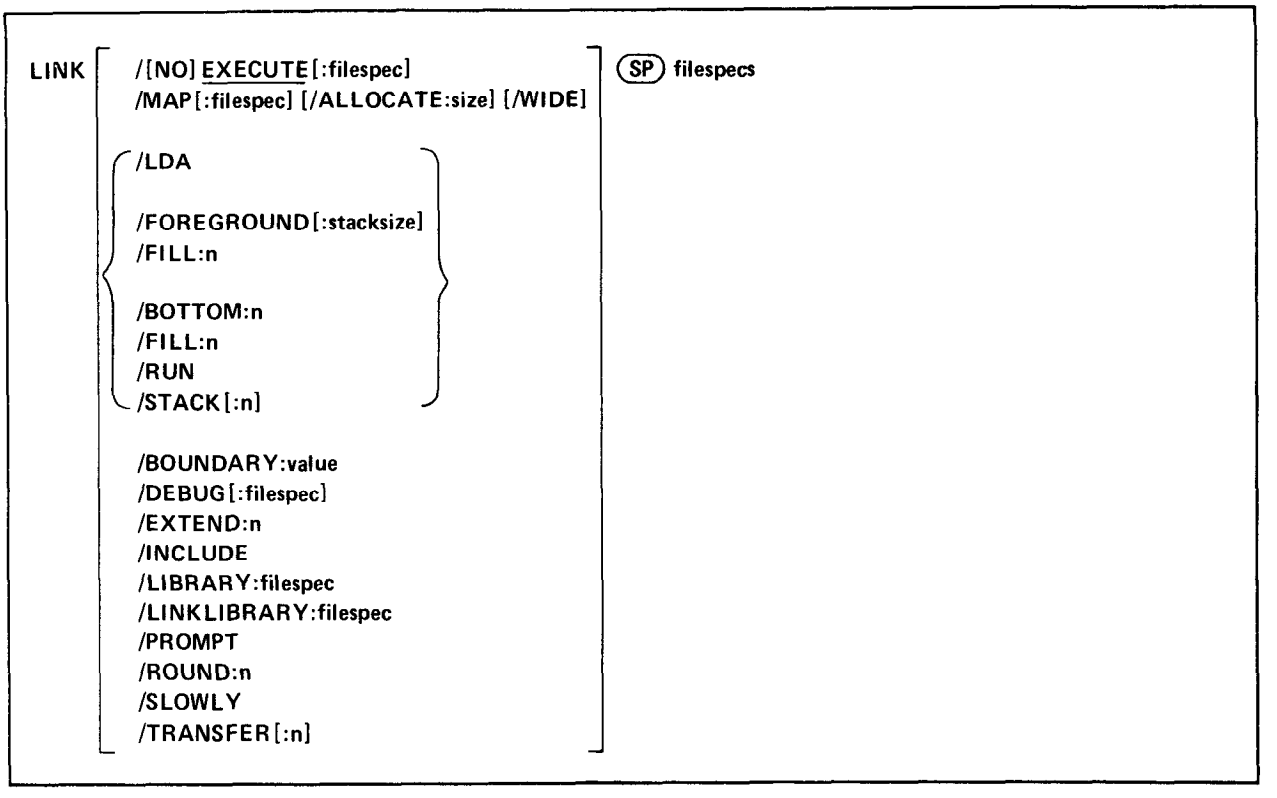
The following example combines several options.

```
 LIBRARY/LIST:TT:/REMOVE/INSERT NEWLIB LIB2/REPLACE,LIB3
Global? SQRT
Global?
RT-11 LIBRARIAN V03.05  FRI 15-JUL-77 00:08:37
NEWLIB                  FRI 15-JUL-77 00:08:35

MODULE          GLOBALS         GLOBALS         GLOBALS

                COS             SIN
                DATAN           DATAN2
                ATAN            ATAN2
                DCOS            DSIN
```

The command executes in the following sequence:

1. Removes global SQRT from NEWLIB
2. Replaces any duplicates of the modules in the file LIB2.OBJ
3. Inserts the modules in the file LIB3.OBJ
4. Lists the directory of NEWLIB.OBJ on the terminal.

The LINK command converts object modules produced by an RT-11 supported language processor into a format suitable for loading and execution.



The RT-11 system lets you separately assemble a main program and each of its subroutines without assigning an absolute load address at assembly time. The linker can then process the object modules of the main program and subroutines to relocate each object module and assign absolute addresses. It links the modules by correlating global symbols that are defined in one module and referenced in another, and it creates the initial control block for the linked program. The linker can also create an overlay structure (if you specify the /PROMPT option) and include the necessary run-time overlay handlers and tables. The linker searches libraries you specify to locate unresolved global symbols, and it automatically searches the default system library, SYSLIB.OBJ, to locate any remaining unresolved globals. Finally, the linker produces a load map (if you specify /MAP) that shows the layout of the executable module. Read Chapter 11 for a more detailed explanation of the RT-11 linker.

In the command syntax illustrated above, filespecs represents the object modules to be linked. Each input module should be stored on a random-access device (disk or DECtape); the output device for the load map file can be any RT-11 device. The output for an .LDA file (if you specify /LDA) can also be any RT-11 device, even those that are not block replaceable, such as paper tape.

The default file types are as follows:

Load Module	:	.SAV, .REL(/FOREGROUND), .LDA(/LDA)
Map Output	:	.MAP
Object Module	:	.OBJ

If you specify two or more files to be linked together, separate the files by commas. The system creates an executable file with the same name as the first file in the input list (unless you use /EXECUTE to change it).

The following sections describe the LINK command options and explain how to use them. The last section under this command describes the LINK prompting sequence for commands that combine two or more LINK options.

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

/BOTTOM:n – Use this option to specify the lowest address to be used by the relocatable code in the load module. The argument, n, represents a six-digit unsigned even octal number. If you do not use this option, the linker positions the load module so that the lowest address is location 1000 (octal). This option is illegal for foreground links.

/BOUNDARY:value – Use the /BOUNDARY option to start a specific program section on a particular address boundary. The system generates a whole number multiple of the value you specify for the starting address of the program section. The argument, value, must be a power of 2. The system extends the size of the previous program section to accommodate the new starting address for the specific section. When you have entered the complete LINK command, the system prompts you for the name of the section whose starting address you need to modify. The prompt is:

```
Boundary section?
```

Respond with the appropriate program section name. Terminate your response with a carriage return.

/DEBUG[:filespec] – Use this option to link ODT (on-line debugging technique, described in Chapter 16) with your program to help you debug it. If you supply the name of another debugging program, the system links the debugger you specify with your program. The system links the debugger low in memory relative to your program.

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

```
.LINK MYPROG

.LINK/EXECUTE MYPROG
```

Both commands link MYPROG.OBJ and produce MYPROG.SAV as a result. The /EXECUTE option has different meanings when it follows the command and when it follows the file specification. The following command creates an executable file called PROG1.SAV on device RK1:.

```
.LINK/EXECUTE:RK1:  PROG1,PROG2
```

The next command creates an executable file called MYPROG.SAV on device DK:.

```
.LINK RTN1,RTN2,MYPROG/EXECUTE
```

/NOEXECUTE – Use this option to suppress creation of an executable file.

/EXTEND:n – This option allows you to extend a program section to a specific octal value, n. The resultant program section size is equal to or greater than the value you specify, depending on the space the object code actually requires. When you have entered the complete LINK command, the system prompts you for the name of the program section you need to extend. The prompt is:

```
Extend section?
```

Respond with the appropriate program section name. Terminate your response with a carriage return.

/FILL:n – Use this option to initialize unused locations in the load module and place a specific value in those locations. The argument, n, represents the octal value to place in the unused locations. Note that the linker automatically initializes unused locations in the load module to 0; use this option to place another value in those locations. This option can be useful in eliminating random results that occur when a program references uninitialized memory by mistake. It can also help you determine which locations have been modified by the program and which are left unchanged.

/FOREGROUND[:stacksize] – This option produces an executable file in relocatable (.REL) format for use as a foreground job under the FB or XM monitor. You cannot use .REL files in the single-job system. This option assigns the default file type .REL to the executable file. The argument, stacksize, represents the number of bytes of stack space to allocate for the foreground job. The value you supply is interpreted as an octal number; specify an even number. Follow n with a decimal point (n.) to represent a decimal number. The default value is 128 (decimal) bytes of stack space.

/INCLUDE – This option lets you take global symbols from any library and include them in the linked memory image. When you have entered the complete LINK command, the system prompts you for a list of global symbols to include in the load module. The prompt is:

```
Library search?
```

Respond by typing the global symbols to be included in the load module. Type a carriage return after each global symbol. Type a carriage return on a line by itself to terminate the list. This provides a method for forcing modules (that are not called by other modules) to be loaded from the library.

/LDA – This option produces an executable file in LDA format. The LDA-format file can be output to any device, including those that are not block-replaceable, such as the paper tape punch or cassette. This option assigns the default file type .LDA to the executable file. This option is useful for files that you need to load with the Absolute Binary Loader.

/LIBRARY – This option is the same as /LINKLIBRARY. It is included for compatibility with other systems.

/LINKLIBRARY:filespec – You can use this option to include the library file you specify as an object module library in the linking operation. This option is not necessary because the system automatically recognizes library files in the linking operation; it is provided for compatibility with the EXECUTE command.

/MAP[:filespec] – You must specify this option to produce a load map listing. The /MAP option has different meanings depending on where you put it in the command line.

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

```
.LINK/MAP:TT:  MYPROG
```

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

```
.LINK/MAP:RK3:  MYPROG
```

If the /MAP option contains a name and file type to override the default of .LST, the system generates a listing with that name. The following command, for example, links PROG1 and PROG2, producing a map listing file called MAP.OUT on device DK:.

```
.LINK/MAP:MAP.OUT PROG1,PROG2
```

Another way to specify /MAP is to type it after the file specification to which it applies. To link a file and produce a map listing file with the same name, use a command similar to this one.

```
.LINK PROG1,PROG2/EXECUTE/MAP
```

The command shown above links PROG1 and PROG2, producing files PROG2.SAV and PROG2.MAP. If you specify a file name on a /MAP option following a file specification in the command line, it has the same meaning as when it follows the command.

/PROMPT – Use this option to enter additional lines of input. The system continues to accept lines of linker input until you enter two slashes (//). Chapter 11 describes the commands you can enter directly to the linker. The /PROMPT option also gives you a convenient way to create an overlaid program from an indirect file. The file HERB.COM contains these lines:

```
A/PROMPT
SUB1/O:1
SUB2/O:1
SUB3,SUB4/O:1
//
```

The following command produces an executable file, DK:HERB.SAV, and a link map on the printer.

```
.LINK/MAP @HERB
```

/ROUND:n – This option rounds up the section you specify so that the size of the root segment is a whole number multiple of the value, n, you supply. The argument, n, must be a power of 2. When you have entered the complete LINK command, the system prompts you for the name of the section that you need to round. The prompt is:

```
Round section?
```

Respond with the appropriate program section name. Terminate your response with a carriage return.

/RUN – Use this option to initiate execution of the resultant .SAV file. This option is valid for background jobs only.

/SLOWLY – This option instructs the system to allow the largest possible memory area for the link symbol table at the expense of making the link process slower. Use this option only if an attempt to link a program failed because of symbol table overflow.

/STACK[:n] – This option lets you modify the stack address. This address, location 42, is the address that contains the value for the stack pointer. When your program executes, the stack pointer (SP) is automatically set to the contents of location 42. The argument, n, is an even, unsigned six-digit octal number that defines the stack address. When you have entered the complete LINK command, the system prints the following prompt message if you did not already specify a numeric value for n.

```
Stack symbol?
```

Respond with the global symbol whose value is the stack address. You cannot specify a number at this point. Terminate your response with a carriage return. If you specify a nonexistent symbol, the system prints an error message. It then sets the stack address to 1000 (for memory image files) or to the bottom address if you used /BOTTOM.

/TRANSFER[:n] – The transfer address is the address at which a program starts when you initiate execution with R, RUN, or FRUN. The /TRANSFER option lets you specify the start address of the load module. The argument, n, is an even, unsigned six-digit octal number that defines the transfer address. When you have entered the complete LINK command, the system prints the following prompt message if you did not already specify a numeric value for n:

```
Transfer symbol?
```

Respond with the global symbol whose value is the transfer address. You cannot specify a number at this point. Terminate your response with a carriage return. If you specify a nonexistent symbol, an error message prints and the linker sets the transfer address to 1 so the system cannot execute the program. If the transfer address you specify is odd, the program does not execute after loading and control returns to the monitor.

/WIDE – Use this option with /MAP to produce a wide load map listing. Normally, the listing is wide enough for three GLOBAL VALUE columns, which is suitable for paper with 72 or 80 columns. The /WIDE option produces a listing that is six GLOBAL VALUE columns wide, which is ideal for a 132-column page.

This section describes the prompting sequence that occurs when you combine the LINK options. Table 4-9 lists the sequence in which the system prompts you for additional information.

Table 4-9 LINK Prompting Sequence

Option	Prompt
/TRANSFER	Transfer symbol?
/STACK	Stack symbol?
/EXTEND:n	Extend section?
/BOUNDARY:value	Boundary section?
/ROUND:n	Round section?
/INCLUDE	Library search?

If you combine any of the options listed in Table 4-9, the system prompts you for information in the sequence shown in the table. Note that the Library search? prompt is always last. This is the only prompt that accepts more than one line as a response. For all the prompts, terminate your response with a carriage return. Terminate your list of responses to the Library search? prompt by placing a carriage return on a line by itself. Note that if the command lines are in an indirect file and the system encounters an end-of-file before all the prompting information has been supplied, it prints the prompt messages on the terminal.

The LOAD command makes a device handler resident in memory for use with BATCH or foreground/background jobs.

LOAD (SP) device[=jobtype] [, . . . device[=jobtype]]

In the command syntax shown above, *device* represents the device handler to be made resident; *jobtype*, which can have the values B or F, assigns the device handler to the background or foreground job, respectively. The jobtype specification is invalid with the SJ monitor.

The LOAD command helps control system execution by bringing a device handler into memory and optionally allocating the device to a job. The system allocates memory for the handler as needed. Before you use a device in a foreground program with the FB monitor, or any device at all with the XM monitor, you must first load the device handler. A device can be owned exclusively by either the foreground or background job. (Note that BATCH, if running, is considered to be a background job under the FB and XM monitors.) This exclusivity prevents the input and output of two different jobs from being intermixed on the same non-file-structured device. In the following example, magtape belongs to the background job while DECtape is available for use by either the background or foreground job; the line printer is owned by the foreground job. All three handlers are made resident in memory.

```
.LOAD DT:,MT:=B,LP:=F
```

Different units of the same random-access device controller can be owned by different jobs. Thus, for example, DT1: can belong to the background job while DT5: can belong to the foreground job. If no ownership is indicated, the device is available for public use. To change ownership of a device, use another LOAD command. It is not necessary to first unload the device. For example, if the line printer has been loaded into memory and assigned to the foreground job as in the example above, the following command reassigns it to the background job without unloading the handler first.

```
.LOAD LP:=B
```

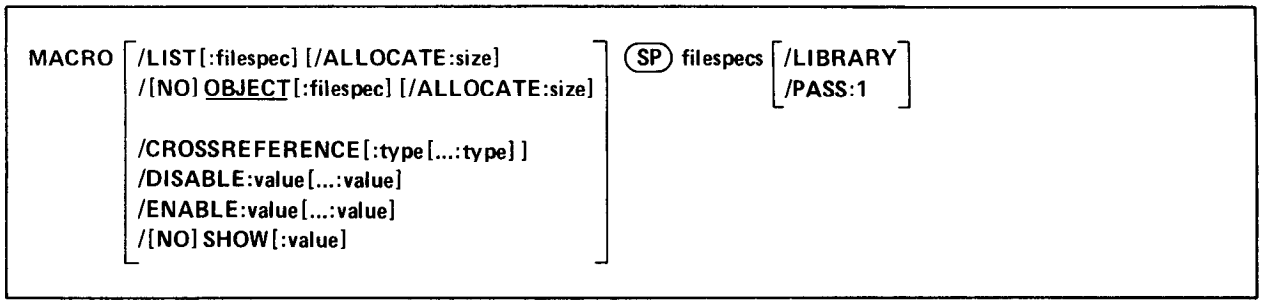
Note, however, that if you interrupt an operation that involves magtape or cassette, you must unload (with the UNLOAD command) then reload the appropriate device handler (MM, MT, or CT).

You cannot assign ownership of the system unit (the unit you bootstrapped) of a system device, and any attempt to do so is ignored. You can, however, assign ownership of other units of the same type as the system device. LOAD is valid for use with user-assigned names. For example:

```
.ASSIGN RK1: XY
.LOAD XY:=F
```

If you are using the diskette monitor, loading the necessary device handlers into memory can improve system performance since no handlers need to be loaded dynamically from the diskette. Use the SHOW DEVICES command to display on the terminal the status of device handler and device ownership.

The MACRO command invokes the MACRO assembler to assemble one or more source files.



In the command syntax shown above, filespecs represents one or more files to be included in the assembly. If you omit a file type for an input file, the system assumes .MAC. Output default file types are .LST for listing files and .OBJ for object files.

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

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

You can enter the MACRO command as one line, or you can rely on the system to prompt you for information. The MACRO command prompt is Files? for the input specification. The system prints on the terminal the number of errors MACRO detects during an assembly, as this printout shows:

```
.MACRO/CROSSREFERENCE PROG1+PROG2/LIST/OBJECT
ERRORS DETECTED:  0
```

Chapter 10 and the *PDP-11 MACRO Language Reference Manual* contain more detailed information about using MACRO. The following sections describe the options you can use with the MACRO command.

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

/CROSSREFERENCE:type[. . . :type] – Use this option to generate a symbol cross-reference section in the listing. This information is useful for program maintenance and debugging. Note that the system does not generate a listing by default. You must also specify /LIST in the command line to get a cross-reference listing. The argument, type, represents a one-character code that indicates which sections of the cross-reference listing the assembler should include. Table 4-10 summarizes the valid arguments and their meanings.

/DISABLE:value[. . . :value] – Use this option to specify a MACRO .DSABL directive. See Section 6.2 of the *PDP-11 MACRO Language Reference Manual* for a description of the directive and a list of all legal values. Table 4-11 summarizes the arguments and their meaning.

Table 4-10 Cross-reference Sections

Argument	Section Type
S	User-defined symbols
R	Register symbols
M	Macro symbolic names
P	Permanent symbols (instructions, directives)
C	Control sections (.CSECT and .PSECT symbolic names)
E	Error codes
no argument	Equivalent to :S:M:E

Table 4-11 .DSABL and .ENABL Directive Summary

Argument	Default	Enables or Disables
ABS	disable	Absolute binary output
AMA	disable	Assembles all absolute addresses as relative addresses
CDR	disable	Treats source columns 73 and greater as comments
FPT	disable	Floating point truncation
GBL	disable	Treats undefined symbols as globals
LC	disable	Accepts lower case ASCII input
LSB	disable	Local symbol block
PNC	enable	Binary output
REG	enable	Mnemonic definitions of registers

/ENABLE:value[. . . :value] – Use this option to specify a MACRO .ENABL directive. See Section 6.2 of the *PDP-11 MACRO Language Reference Manual* for a description of the directive and a list of all legal values. Table 4-11 summarizes the arguments and their meaning.

/LIBRARY – This option identifies the file it qualifies as a library file; use it only after a macro library file specification in the command line. The MACRO assembler looks first to the library file or files you specify and then to the system library, SYSMAC.SML, to satisfy references (made with the .MCALL directive) from MACRO programs. In the example below, the command string includes two user libraries.

```
.MACRO MYLIB1/LIBRARY+A+MYLIB2/LIBRARY+B
```

When MACRO assembles file A, it looks first to the library, MYLIB1.MAC, and then to SYSMAC.SML to satisfy .MCALL references. When it assembles file B, MACRO searches MYLIB2.MAC, MYLIB1.MAC, and then SYSMAC.SML, in that order, to satisfy references.

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

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

```
.MACRO/LIST:TT: A
```

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

```
.MACRO/LIST:RK3: A
```

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

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

You cannot use a command like the next one. In this example, the second listing file would replace the first one and, therefore, cause an error.

```
.MACRO/LIST:FILE2 A,B
```

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

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

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

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

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

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

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

```
.MACRO A/LIST,B
```

This command assembles A.MAC, producing A.OBJ and A.LST. It also assembles B.MAC, producing B.OBJ. However, it does not produce any listing file for the assembly of B.MAC.

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

```
.MACRO A
```

```
.MACRO/OBJECT A
```

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

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

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

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

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

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

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

/PASS:1 – Use this option on a prefix macro file to process that file during pass-1 of the assembly only. This option is useful when you assemble a source program together with a prefix file (one that contains only macro definitions), since these definitions do not need to be redefined in pass-2 of the assembly. The following command assembles a prefix file and a source file together, producing files PROG1.OBJ and PROG1.LST.

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

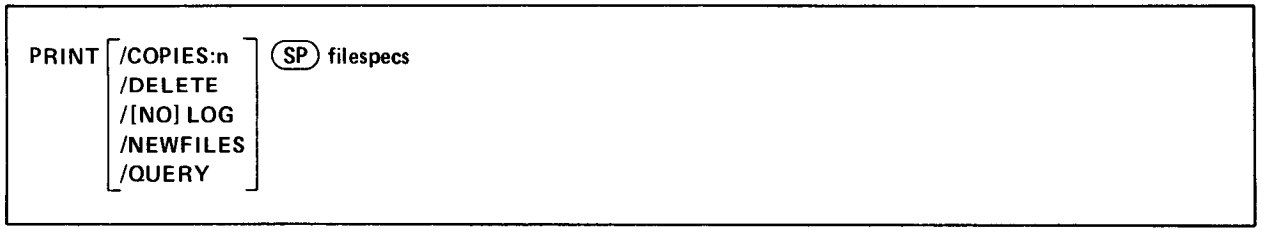
/SHOW:value – Use this option to specify any MACRO .LIST directive. Section 6.1.1 of the *PDP-11 MACRO Language Reference Manual* explains how to use these directives. Table 4-12 summarizes the valid arguments and their meaning.

Table 4-12 .LIST and .NLIST Directive Summary

Argument	Default	Controls listing of
SEQ	list	Source line sequence numbers
LOC	list	Location counter
BIN	list	Generated binary code
BEX	list	Binary extensions
SRC	list	Source code
COM	list	Comments
MD	list	Macro definitions, repeat range expansions
MC	list	Macro calls, repeat range expansions
ME	nolist	Macro expansions
MEB	nolist	Macro expansion binary code
CND	list	Unsatisfied conditionals, .IF and .ENDC statements
LD	nolist	Listing directives with no arguments
TOC	list	Table of Contents
TTM	terminal mode	Listing output format
SYM	list	Symbol table

/NOSHOW:value – Use this option to specify any MACRO .NLIST directive. Section 6.1.1 of the *PDP-11 MACRO Language Reference Manual* explains how to use these directives. Table 4-12 summarizes the valid arguments and their meaning.

The PRINT command lists the contents of one or more files on the line printer.



In the command syntax illustrated above, filespecs represents the file or files to be printed. You can explicitly specify up to six files as input to the PRINT command. The system prints the files in the order in which you specify them in the command line. You can also use wildcards in the file specification. In this case, the system lists the files in the order in which they occur in the directory of the device you specify. If you specify more than one file, separate the files by commas. If you omit the file type for a file specification, the system assumes .LST. You can specify the entire command on one line, or you can rely on the system to prompt you for information. The PRINT command prompt is Files?. Note that if the output device is an LP05, you must terminate the file with a line feed, form feed, or carriage return.

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

/COPIES:n – Use this option to print more than one copy of the file. The meaningful range of values for the decimal argument, n, is from 2 to 32 (1 is the default). The following command, for example, prints three copies of the file REPORT.LST on the line printer.

```
.PRINT/COPIES:3 REPORT
```

/DELETE – Use this option to delete a file after it prints on the line printer. This option must appear following the command in the command line. The PRINT/DELETE operation does not ask you for confirmation before it executes. You must use /QUERY for this function. The following example prints a BASIC program on the line printer, then deletes it from DX1:.

```
.PRINT/DELETE DX1:PROG1.BAS
```

/LOG – This option lists on the terminal the names of the files that are printed by the current command. Normally the system prints a log only if there is a wildcard in the file specification. If you specify /QUERY, the query messages replace the log, unless you specifically type /LOG/QUERY in the command line. The following example shows a PRINT command and the resulting log.

```
.PRINT/LOG/DELETE REPORT
 Files copies/deleted:
DK:REPORT.LST to LP:
```

/NOLOG – This option prevents a list of the files that were printed from typing out on the terminal. You can use this option to suppress the log when you use a wildcard in the file specification.

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

```
.PRINT/NEWFILES *.LST
 Files copied:
DK:OUTFIL.LST    to LP:
DK:REPORT.LST    to LP:
```

/QUERY – If you use this option, the system requests confirmation from you before it performs the operation. /QUERY is particularly useful on operations that involve wildcards, when you may not be completely sure which files the system selected for an operation. Note that if you specify /QUERY in a PRINT command line that also contains a wildcard in the file specification, the confirmation messages that print on the terminal replace the log messages that would normally appear. You must respond to a query message by typing Y (or anything that begins with Y) and a carriage return to initiate execution of a particular operation. The system interprets any other response to mean NO; it does not perform the specific operation. The following example uses /QUERY.

```
.PRINT/QUERY *.LST
 Files copied:
DK:OUTFIL.LST    to LP:? NO
DK:REPORT.LST    to LP:? Y
```

The R command loads a memory image file from the system device into memory and starts execution.

R (SP) filespec

In the command syntax shown above, filespec represents the program to be executed. The default file type is .SAV. The default device is SY:. The R command is similar to the RUN command except that the file you specify in an R command string must be on the system device (SY:). Use the R command only with background jobs. The following command loads and executes MYPROG.SAV from device SY:.

```
.R MYPROG
```

The REENTER command starts the program at its reentry address (the start address minus two).

REENTER

The REENTER command accepts no options or arguments. REENTER does not clear or reset any memory areas. Use it to avoid reloading the same program for repetitive execution. You can use REENTER to return to a system program or to any program that allows for a REENTER after the program terminates. You can also use REENTER after you have used two CTRL/Cs to interrupt those programs.

If you issue the REENTER command and it is not valid for a program, the message ?KMON-F-Illegal command prints. You must start that program with an R or RUN command.

In the following example the directory program (DIR) lists the directory of DK: on the line printer. Two CTRL/Cs interrupt the listing and return to the monitor. REENTER starts DIR at its reentry address and DIR prompts for a line of input.

```
.R DIR
*LP:=DK:*.*
^C
^C
.
.REENTER
*
```

Note in the example above that using REENTER does not continue the directory listing where it was interrupted.

The REMOVE command removes a device from the system tables.

REMOVE (SP) device[, . . . device]

In the command syntax shown above, device represents the device to remove from the system tables. The REMOVE command accepts no options. You can enter the REMOVE command on one line, or you can rely on the system to prompt you for information. The REMOVE command prompt is Device?.

Using the REMOVE command does not change the monitor disk image; it only modifies the system tables of the monitor currently in core. This allows you to configure a special system for a single session at the computer without having to reconfigure to return to your standard device configuration. Bootstrapping the system device restores the original device configuration. To permanently REMOVE a device, include the REMOVE command in the standard system startup indirect command file.

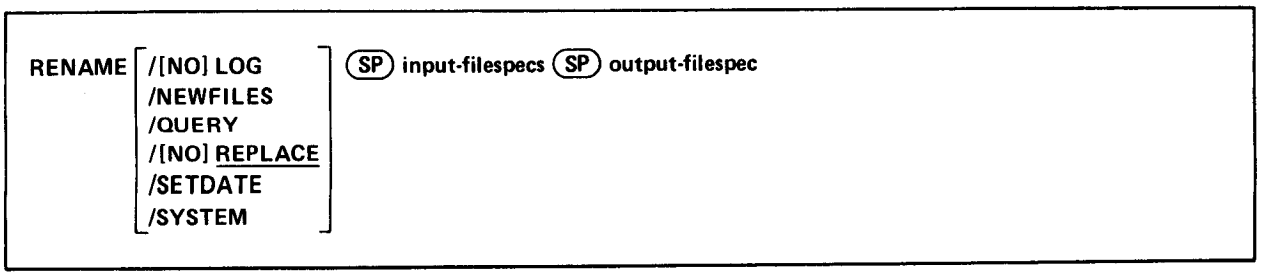
You cannot remove the following system devices: SY (the handler for the system device), BA (the BATCH handler), and TT (the terminal handler). You can use the INSTALL command to install a new device after using the REMOVE command to remove a device (thus creating a free device slot).

The following command removes the line printer handler and the card reader handler from the system. Note that the colons (:) are optional.

```
.REMOVE  LP:,CR:
```

Use the SHOW DEVICES command to display on the terminal a list of devices that are currently available on your system.

The RENAME command assigns a new name to an existing file.



In the command syntax illustrated above, input-filespecs represents the files to be renamed, and output-filespec represents the new name. You can specify up to six input files, but only one output file. Note that the device specification must be the same for input and output; you cannot rename a file from one device to another. If a file exists with the same name and file type as the output file you specify, the system deletes the existing file unless you use the /NOREPLACE option to prevent this.

The system has a special way of handling system (.SYS) files and files that cover bad blocks (.BAD) files. So that you do not rename system files by accident when you use a wildcard in the file specification, the system requires you to use the /SYSTEM option when you need to rename system files. To rename a .BAD file, you must specify it by explicitly giving its file name and file type. Since .BAD files cover bad blocks on a device, you usually do not need to rename or otherwise manipulate these files.

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

/LOG – This option lists on the terminal the files that were renamed by the current command. Normally, the system prints a log only if there is a wildcard in the file specification. If you specify /QUERY, the query messages replace the log (unless you specifically type /LOG/QUERY in the command line).

This example demonstrates logging.

```
.RENAME DX0:A%%.MAC DX0:*.FOR
 Files renamed:
DX0:ABC.MAC     to DX0:ABC.FOR
DX0:AAF.MAC     to DX0:AAF.FOR
```

/NOLOG – This option prevents a list of the files that are renamed from appearing on the terminal.

/NEWFILES – Use this option in the command line if you want to rename only those files that have the current date. This is a convenient way to access all new files after a session at the computer.

/QUERY – If you use this option, the system requests confirmation from you before it performs the operation. /QUERY is particularly useful on operations that involve wildcards, when you may not be completely sure which files the system selected for the operation. Note that if you specify /QUERY in a command line that also contains a wildcard in the file specification, the confirmation messages that print on the terminal replace the log messages that would normally appear. You must respond to a query message by typing Y (or anything that begins with Y) and a carriage return to initiate execution of a particular operation. The system interprets any other response to mean NO; it does not perform the specific operation. This example demonstrates querying.

```
.RENAME/QUERY DX0:(PIP1.SAV PIP.SAV)
 Files renamed:
DX0:PIP1.SAV    to DX0:PIP.SAV    ? Y
```

/REPLACE – This is the default mode of operation for the RENAME command. If a file exists with the same name as the file you specify for output, the system deletes that duplicate file when it performs the rename operation.

/NOREPLACE – This option prevents execution of the rename operation if a file with the same name as the output file you specify already exists on the same device. The following example uses /NOREPLACE. In this case, the output file already exists and no action occurs.

```
.RENAME/NOREPLACE DX0:TEST.SAV DX0:DUP.SAV
?PIP-W-Output file found, no operation performed DX0:TEST.SAV
```

/SETDATE – This option causes the system to put the current date on all files it renames, unless the current system date is not set. Normally, the system preserves the existing file creation date when it renames a file. The following example renames files and changes their dates.

```
.RENAME/SETDATE DX0:(*.FOR *.OLD)
 Files renamed:
DX0:ABC.FOR     to DX0:ABC.OLD
DX0:AAF.FOR     to DX0:AAF.OLD
DX0:MERGE.FOR   to DX0:MERGE.OLD
```

/SYSTEM – Use this option if you need to rename system (.SYS) files. If you omit this option, the system files are excluded from the rename operation and a message is printed on the terminal to remind you of this. This example renames MM.SYS to MX.SYS.

```
.RENAME/SYSTEM DX0:MM.SYS DX0:MX.SYS
```

The RESET command resets several background system tables and does a general clean-up of the background area.

RESET

The RESET command accepts no options or arguments. The RESET command causes the system to purge all open input/output channels, initialize the user program memory area, and unload any device handlers that were not explicitly made resident with the LOAD command. It also disables CTRL/O, clears locations 40-53, and resets the KMON (keyboard monitor) stack pointer. Use RESET before you execute a program if a device or the monitor needs reinitialization, or when you need to discard the results of previously issued GET commands. The RESET command had no effect on the foreground job. The following example uses the RESET command before running a program.

```
.RESET
.R MYPROG
```

The RESUME command continues execution of the foreground job at the point the SUSPEND command was issued.

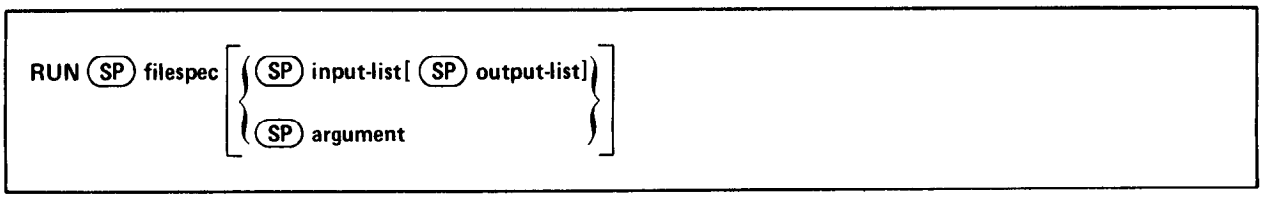
RESUME

No arguments or options are permitted with the RESUME command. When you issue the RESUME command, the foreground job enters any completion routines that were scheduled while the job was suspended. Note that RESUME is valid only with the FB and XM monitors. The following command resumes execution of the foreground job that is currently suspended.

```
.RESUME
```

You can also use the RESUME command to execute a foreground job that you start with FRUN using /P.

The RUN command loads a memory image file into memory and starts execution.



In the command syntax illustrated above, filespec represents the program to execute. The system assumes a .SAV file type for the executable file, which can reside on any RT-11 block-replaceable device. The default device is DK:. The RUN command automatically loads the device handler for the device you specify if it is not already resident. This eliminates the need to explicitly load a device handler when you run an overlaid program from a device other than the system device. The RUN command executes only those programs that have been linked to run as background jobs. You cannot use RUN on a virtual job that executes under the XM monitor. The following command, for example, executes MYPROG.SAV, which is stored on device DX1:.

```
.RUN DX1:MYPROG
```

You can also specify in the RUN command an argument to pass to the program, or a list of input and output specifications. This allows you to specify a line of input for a user program or for a system utility program (which accepts file specifications in the special syntax described in Chapter 6). The system automatically converts the input-list and the output-list you specify into a format that the CSI (Command String Interpreter) accepts. For example, to execute the directory program (DIR) and obtain a complete listing of the directory of DX1: on the printer, you can use the following command.

```
.RUN DIR DX1:*.* LP:/E
.
```

This command has the same effect as the following lines.

```
.RUN DIR
*LP:/E=DX1:*.*
*^C
.
```

Note that when you use either an argument or an input-list and output-list with RUN, control returns to the monitor when the program completes.

The SAVE command writes memory areas in memory image format to the file and device that you specify.

SAVE (SP) filespec[(SP) parameters]

In the command syntax shown above, filespec represents the file to be saved on a block-replaceable device. If you do not specify a file type, the system uses .SAV. The parameters represent memory locations to be saved.

Parameters are of the form:

address[-address(2)] [,address(3)[-address(n)]]

where

address — is an octal value representing a specific block of memory locations to be saved. If you specify more than one address, each address must be higher than the previous one.

RT-11 transfers memory in 256-word blocks beginning on boundaries that are multiples of 256 (decimal). If the locations you specify make a block that is less than 256 words, the system saves additional words to make a 256-word block.

The system saves memory from location 0 to the highest memory address specified by the parameter list or to the program high limit (location 50 in the system communication area). Initially, the system gives the start address and the JSW (Job Status Word) the default value 0 and sets the stack to 1000. If you want to change these or any of the following addresses, you can use the Deposit command to alter them and the SAVE command to save the correct areas.

AREA	LOCATION
Start address	40
Stack	42
JSW	44
USR address	46
High address	50
Fill characters	56

If you change the values of the addresses, it is your responsibility to reset them to their default values. For more information concerning these addresses refer to the *RT-11 Advanced Programmer's Guide*. Note that the SAVE command does not write the overlay segments of programs; it saves only the root segment.

The following command saves location 10000 through 11777 and 14000 through 14777. It stores the contents of these locations in the file FILE1.SAV on device DK:.

```
.SAVE FILE1 10000-11000,14000-14100
```

The next example sets the reenter bit in the JSW and saves locations 1000 through 5777 in file PRAM.SAV on device SY:.

```
.D 44=2000
.SAVE SY:PRAM 1000-5777
```

The SET command changes device handler characteristics and certain system configuration parameters.

```
SET (SP) { physical-device-name } (SP) condition
         { item               }
```

In the command syntax illustrated above, physical-device-name represents the device handler whose characteristics you need to modify.

See Table 3-1 for a list of the standard RT-11 permanent device names. The argument, item, represents a system parameter that you need to modify. The system items you can change include error handling (SET ERROR) and wildcard handling (SET WILDCARDS). Table 4-13 lists the devices and items you can modify as well as the valid conditions for these devices and items. If you set more than one condition for a device, separate the conditions by commas. With the exception of the SET TT, SET USR, and SET item commands, the SET command locates the file SY:device.SYS and permanently modifies it. The SET commands are valid for all three RT-11 monitors unless otherwise specified. They permanently modify the device handlers (except where noted); this means that the conditions remain set even across a reboot. For those SET commands that do not permanently modify the device handlers, the conditions return to the default setting after a reboot. To make these settings appear permanent, include the appropriate SET commands in your system's startup indirect command file (see Section 4.3.3). The command you enter must be completely valid for the modification to take place. If a handler is already loaded when you issue a SET command for it, you must unload the handler and install a fresh copy from the system device for the modification to have an effect on execution. Note that the colon (:) after each device name is optional.

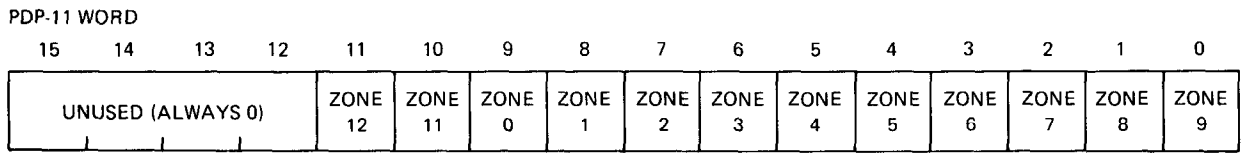


Figure 4-2 Format of a 12-bit Binary Number

Table 4-13 SET Device Conditions

Device or Item	Condition	Action
CR:	CODE=n	Modifies the card reader handler to use either the DEC 026 or DEC 029 card codes. The argument, n, must be either 26 or 29. The default value is 29.
CR:	CRLF	Appends a carriage return/line feed combination to each card image. This is the normal mode.
CR:	NOCRLF	Transfers each card image without appending a carriage return/line feed combination. The default is CRLF.
CR:	HANG	Waits for you to make a correction if the reader is not ready at the start of a transfer. This is the normal mode.

(Continued on next page)

Table 4-13 (Cont.) SET Device Conditions

Device or Item	Condition	Action
CR:	NOHANG	Generates an immediate error if the device is not ready at the start of a transfer. The handler waits (regardless of how the condition is set) if the reader becomes not ready during a transfer (i.e., the input hopper is empty, but an end-of-file card has not been read). The default is HANG.
CR:	IMAGE	Causes each card column to be stored as a 12-bit binary number, one column per word. The CODE option has no effect in IMAGE mode. Figure 4-2 illustrates the format of the 12-bit binary number. This format allows the system to read binary card images. It is especially useful if you use a special encoding of punch combinations. Mark-sense cards can be read in this mode. The default is NOIMAGE.
CR:	NOIMAGE	Allows the normal translation (as specified by the CODE option) to take place. The system packs data one column per byte. It translates invalid punch combinations into the error character, ASCII backslash (\), which is octal code 134. This is the normal mode.
CR:	TRIM	Removes trailing blanks from each card that the system reads. You should not use TRIM and NOCRLF together because card boundaries become difficult to read. TRIM is the normal mode.
CR:	NOTRIM	Transfers a full 80 characters per card. The default is TRIM.
CT:	RAW	Performs a read-after-write check for every record written. It retries if an output error occurs. If three retries fail, the system indicates an output error. The default is NORAW.
CT:	NORAW	Writes every record directly without reading it back for verification. This setting significantly increases transfer rates at the risk of increased error rates. This is the normal mode.
EDIT	EDIT	Invokes the text editor EDIT with the keyboard monitor EDIT command. This is the normal mode. The system returns to this condition after a reboot.
EDIT	TECO	Invokes the text editor TECO with the keyboard monitor EDIT command. The default is EDIT. The system returns to that condition after a reboot.
ERROR	ERROR	Causes indirect command files and keyboard monitor commands that perform multiple operations (such as EXECUTE, which combines assembling, linking, and running) to abort if errors or severe errors occur. An example of an error is an undefined symbol in an assembly. An example of a severe error is a device that is write-locked when the system attempts to write to it. If either condition occurs, the indirect command file or keyboard monitor command aborts the next time the monitor get control of the system. This is the normal setting. The system returns to this condition after a reboot.

(Continued on next page)

Table 4-13 (Cont.) SET Device Conditions

Device or Item	Condition	Action
ERROR	NONE	Allows indirect command files and keyboard monitor commands to continue to execute even though they contain significant errors. Most monitor fatal errors still cause the indirect command file or keyboard monitor command to abort. See SET ERROR ERROR. SET ERROR ERROR is the default setting. The system returns to that condition after a reboot.
ERROR	SEVERE	Causes indirect command files and keyboard monitor commands to abort if severe errors occur. See SET ERROR ERROR. SET ERROR ERROR is the default setting. The system returns to that condition after a reboot.
ERROR	WARNING	Causes indirect command files and keyboard monitor commands to abort if warnings, errors, or severe errors occur. See SET ERROR ERROR. SET ERROR ERROR is the default setting. The system returns to that condition after a reboot.
LP:	CR	Sends carriage returns to the printer. To allow overstriking on the printer, use this condition for any FORTRAN program that uses formatted input and output. Use CR also for any LS11 or LP05 line printer to prevent loss of the last line in the buffer. This is the normal mode.
LP:	NOCR	Prevents the system from sending carriage returns to the printer. This setting produces a significant increase in printing speed on LP11 printers. The line printer controller causes a line feed to perform the functions of a carriage return. The default is CR.
LP:	CTRL	Passes all characters, including nonprinting control characters, to the printer. Use this condition to pass the bell character to the LA180 printing terminal. You can use this mode for LS11 line printers. (Other line printers print a space for a control character.) The default is NOCTRL.
LP:	NOCTRL	Ignores non-printing control characters. This is the normal mode.
LP:	FORM0	Issues a form feed before a request to print blocks 0. This is the normal mode.
LP:	NOFORM0	Turns off FORM0 mode. The default is FORM0.
LP:	HANG	Waits for you to make a correction if the line printer is not ready or becomes not ready during printing. If you expect output from the line printer and the system does not respond or appears to be idle, check to see if the line printer is powered on and ready to print. This is the normal mode.

(Continued on next page)

Table 4-13 (Cont.) SET Device Conditions

Device or Item	Condition	Action
LP:	NOHANG	Generates an immediate error if the line printer is not ready. The default is NOHANG.
LP:	LC	Allows the system to send lower case characters to the printer. Use this condition if your printer has a lower case character set. The default is NOLC.
LP:	NOLC	Translates lower case characters to upper case before printing. This is the normal mode.
LP:	TAB	Sends TAB characters to the LA180 line printer. The default is NOTAB.
LP:	NOTAB	Does not send TAB characters to the line printer. This is the normal mode.
LP:	WIDTH=n	Sets the line printer width to n, where n is an integer between 30 and 255, inclusive. The system ignores any characters that print past column n.
MM:	DEFALT=9	Returns to default settings for 9-track tape. The 9-track defaults are: DENSE=809 ODDPAR NODUMP
MM:	DENSE=[800 or 809 or 1600]	Sets density for the 9-track tape handler. Do not alter the density setting within a volume. A density setting of 1600 bits per inch (BPI) automatically sets parity to odd. The valid density settings for 9-track tape are: 800 BPI 1600 BPI
MM:	ODDPAR	Sets parity to odd for 9-track tape. DIGITAL recommends this setting.
MM:	NOODDPAR	Sets parity to even for 9-track tape. DIGITAL does not recommend this setting for normal operation, and provides it only for compatibility with other systems.
MT:	DEFALT=[7 or 9]	Returns to default settings for 7- or 9-track tape. The 7-track defaults are: DENSE=807 ODDPAR DUMP

(Continued on next page)

Table 4-13 (Cont.) SET Device Conditions

Device or Item	Condition	Action
MT: (Cont.)		The 9-track defaults are: DENSE=809 ODDPAR NODUMP
MT:	DENSE=[200 or 556 or 807 or 800 or 809]	Sets density for 7- or 9-track tape. 807 represents 800 BPI for 7-track tape; 800 or 809 represents 800 BPI for 9-track tape. Do not alter the density within a tape volume. You must set density to 807 for 7 track tape if you want dump mode. The valid density settings for 7 and 9 track tape are: 7-track: 200 BPI 556 BPI 800 BPI 800 BPI Dump 9-track: 800 BPI
MT:	DUMP	Writes bytes to 7-track tape. You must also set density to 807.
MT:	ODDPAR	Sets parity to odd for 7- or 9-track tape. DIGITAL recommends this setting.
MT:	NOODDPAR	Sets parity to even for 7- or 9-track tape. DIGITAL does not recommend this setting for normal operation, and provides it only for compatibility with other systems.
TT:	CONSOL=n	Directs the system to use as the console terminal, the terminal whose logical unit number you specify. The default value is 0, which represents the original console terminal. The terminal whose logical unit number you specify must not be currently attached by the foreground job. The system returns to this default after a reboot.
TT:	CRLF	Issues a carriage return/line feed combination on the console terminal whenever you attempt to type past the right margin. You can change the margin with the WIDTH command. This is the normal mode. This setting is not valid for the SJ monitor. The system returns to this condition after a reboot.
TT:	NOCRLF	Takes no special action at the right margin. This setting is not valid for the SJ monitor. The default is CRLF. The system returns to that condition after a reboot.

(Continued on next page)

Table 4-13 (Cont.) SET Device Conditions

Device or Item	Condition	Action
TT:	FB	Treats CTRL/B and CTRL/F as background and foreground program control characters and does not transmit them to your program. This is the normal mode. This setting is not valid for the SJ monitor. The system returns to this condition after a reboot.
TT:	NOFB	Causes CTRL/B and CTRL/F to have no special meaning. Issue SET TT NOFB to KMON, which runs as a background job, to disable all communication with the foreground job. To enable communication with the foreground job, issue the command SET TT FB. This setting is not valid for the SJ monitor. The default is FB. The system returns to that condition after a reboot.
TT:	FORM	Indicates that the console terminal is capable of executing hardware form feeds. This setting is not valid for the SJ monitor.
TT:	NOFORM	Simulates form feeds by generating eight line feeds. This setting is not valid for the SJ monitor. This is the normal mode. The system returns to this condition after a reboot.
TT:	HOLD	Enables the Hold Screen mode of operation for the VT50 terminal. The command has no effect on any other terminal, but it can cause a left square bracket ([) to print. This setting is valid for all monitors. This is the normal mode. The system returns to this condition after a reboot.
TT:	NOHOLD	Disables the Hold Screen mode of operation for the VT50 terminal. The command has no effect on any other terminal, but it can cause a backslash (\) to print. This setting is valid for all monitors. The default is HOLD. The system returns to that condition after a reboot.
TT:	PAGE	Treats CTRL/S and CTRL/Q characters as terminal output hold and unhold flags and does not transmit them to your program. This setting is not valid for the SJ monitor. This is the normal mode. The system returns to this condition after a reboot.
TT:	NOPAGE	Causes CTRL/S and CTRL/Q to have no special meaning. This setting is not valid for the SJ monitor. The default is PAGE. The system returns to that condition after a reboot.
TT:	QUIET	Prevents the system from echoing lines from indirect files. The default is NOQUIET. The system returns to that condition after a reboot.
TT:	NOQUIET	Echoes lines from indirect files. This is the default mode. The system returns to this condition after a reboot.
TT:	SCOPE	Echoes RUBOUT characters as backspace-space-backspace. Use this mode if your console terminal is a VT50, VT05, VT52, VT55, VT61, or if GT ON is in effect. This setting is not valid for the SJ monitor. The default is NOSCOPE. The system returns to that condition after a reboot.

(Continued on next page)

Table 4-13 (Cont.) SET Device Conditions

Device or Item	Condition	Action
TT:	NOSCOPE	Echoes each RUBOUT character as a backslash followed by the character deleted. This is the normal mode. This setting is not valid for the SJ monitor. The system returns to this condition after a reboot.
TT:	TAB	Indicates that the console terminal is capable of executing hardware tabs. This setting is not valid for the SJ monitor. The default is NOTAB. The system returns to that condition after a reboot.
TT:	NOTAB	Simulates tab stops every eight positions. VT05 and VT50 terminals generally have hardware tabs. This setting is not valid for the SJ monitor. This is the normal mode. The system returns to this condition after a reboot.
TT:	WIDTH=n	Sets the terminal width to n, where n is an integer between 30 and 255. The system initially sets the width to 72. This setting is not valid for the SJ monitor. (See SET TT CRLF.) The system returns to width 72 after a reboot.
WILDCARDS	EXPLICIT	Causes the system to recognize file specifications exactly as you type them. If you omit a file name or a file type in a file specification the system does not automatically replace the missing item with an asterisk (*). Wildcards are described in Section 4.2. The default is IMPLICIT. The system returns to that condition after a reboot.
WILDCARDS	IMPLICIT	Causes the system to interpret missing fields in file specifications of certain commands as asterisks (*). Wildcards are described in Section 4.2 of this manual. Table 4-2 shows how the system interprets commands that have missing fields. This is the normal mode. The system returns to this condition after a reboot.
USR	SWAP	Allows the background job to place the USR in a swapping state. This setting is not valid for the XM monitor. This is the normal mode. The system returns to this condition after a reboot.
USR	NOSWAP	Prevents the background job from placing the USR in a swapping state This setting is not valid for the XM monitor. The default is SWAP. The system returns to that condition after a reboot.

The following examples illustrate the SET command. This command allows the system to send lower case characters to the printer:

```
.SET LP: LC
```

The next command sets the system wildcard default to implicit.

```
.SET WILDCARDS IMPLICIT
```

As a result of this command the system inserts an asterisk in place of a missing file name or file type in a file specification for certain commands. See Table 4-2 for a list of these commands.

The SHOW command prints at the terminal all the devices known to the system and any logical names assigned to these devices.

SHOW[(SP) DEVICES]

The devices the system lists are those known by the RT-11 monitor currently running in memory. This list reflects any additions or deletions you have made with the INSTALL and REMOVE commands. The final entry in the listing shows whether the USR is set to SWAP or NOSWAP. The listing also includes additional information about particular devices. The informational messages and their meanings are:

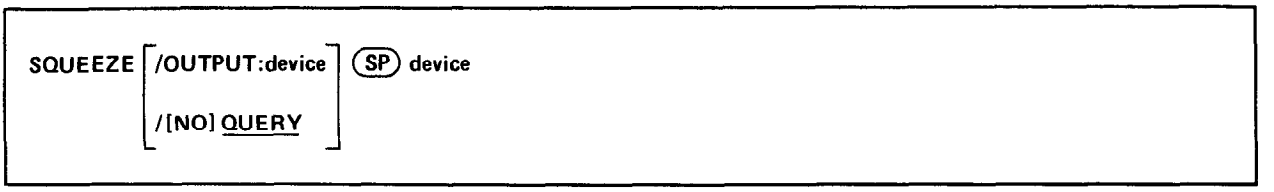
(B) or =B	Indicates that the device or unit is assigned to the background job. (For FB and XM monitors only.)
(F) or =F	Indicates that the device or unit is assigned to the foreground job. (For FB and XM monitors only.)
<FREE>	Shows that the device slot is unused. You can use the INSTALL command to install a device into the free slot. Create a free slot by using the REMOVE command to remove a device.
(LOADED)	Shows that the handler for the device has been loaded into memory with the LOAD command.
(RESIDENT)	Indicates that the handler for the device is included in the resident monitor.
=logical-device-name(1), logical-device-name(2) . . . ,logical-device-name(n)	Shows that the device or unit has been assigned the indicated logical device names with the ASSIGN command.

The following example was created under the FB monitor. It shows the status of all devices known to the system.

```
.SHOW
TT   (Resident)
RK   (Resident)
  RK0    = SY
<Free>
<Free>
DX   (Loaded)
  DX0  (B)
  DX1      = DK
DT
MT   (Loaded=F)
CT
LP         = OUT
<Free>
<Free>
BA
EL
NL
<Free>

USR Swap
```

The listing shows first that TT and RK are resident in memory. The other device handlers known to the system are: DX, DT, MT, CT, LP, BA, EL, and NL. There are five free slots in the table. RK0: has the logical name SY: and DX1: has the logical name DK:. The logical name OUT: is assigned to LP:. The DX handler is loaded and device DX0: belongs to the background job. The MT handler is loaded and belongs to the foreground job. The USR is set to SWAP.

The SQUEEZE command consolidates in a single area all unused blocks on the device you specify.



In the command syntax illustrated above, device represents the disk or DECtape to be compressed. To perform a squeeze operation, the system moves all the files to the beginning of the device you specify, producing a single unused area after the group of files. The squeeze operation does not change the bootstrap blocks of a device. The system prints a confirmation message before it performs the squeeze operation. You must type Y followed by a carriage return to execute the command.

The squeeze operation does not move files with .BAD file types. This feature prevents you from reusing bad blocks that occur on a disk. The system inserts files before and after .BAD files until the space between the last file it moved and the .BAD file is smaller than the next file to be moved.

If you perform a squeeze operation on the system device, the system automatically reboots when the compress operation completes. This reboot takes place in order to prevent system crashes that might occur when the monitor file is moved.

/OUTPUT:filespec – Use this option to transfer all the files from the input device to the output device in compressed format. This operation leaves the input device unchanged. The output device must be an initialized disk or DECtape. (Use the INITIALIZE command to do this.) Note that the system never queries you for confirmation before this operation proceeds. If the output device is not initialized, the system prints an error message and does not execute the command. The following example transfers all the files from RK0: to RK1: in compressed format, leaving RK0: unchanged.

```
.SQUEEZE/OUTPUT:RK1: RK0:
```

/QUERY – This option causes the system to print a confirmation message before it executes a squeeze operation. You must respond by typing a Y followed by a carriage return for execution to proceed. This is the default operation. /QUERY is meaningless with the /OUTPUT option.

/NOQUERY – Use this option to suppress the confirmation message that prints before a squeeze operation executes. The following command compresses all the files on device DT1: and does not query.

```
.SQUEEZE/NOQUERY DT1:
```

The START command initiates execution of the program currently in memory (loaded with the GET command) at the address you specify.

START[(SP) address]

In the command syntax shown above, address is an even octal number representing any 16-bit address. If you omit the address or if you specify 0, the system uses the starting address that is in location 40. If the address you specify does not exist or is invalid for any reason, a trap to location 4 occurs and the monitor prints an error message. Note that this command is valid for background jobs only. The following command loads MYPROG.SAV into memory and begins execution.

```
.GET MYPROG
.START
```

The next example loads MYPROG.SAV and ODT.SAV into memory, and begins execution at ODT's starting address.

```
.GET MYPROG
.GET ODT
.START
 ODT V01.04
*
```

The SUSPEND command stops execution of the foreground job.

SUSPEND

No arguments or options are accepted with this command. The SUSPEND command is not valid for the SJ monitor. The system permits foreground input and output that are already in progress to finish; however, it issues no new input or output requests and enters no completion routines (see the *RT-11 Advanced Programmer's Guide* for a detailed explanation of completion routines). You can continue execution of the job by typing the RESUME command. The following command suspends execution of the foreground job that is currently running.

```
.SUSPEND
```

Use the TIME command to set the time of day or to display the current time of day.

TIME[(SP) hh:mm:ss]

In the command syntax shown above, hh represents hours (from 0 to 23); mm represents minutes (from 0 to 59) and ss represents seconds (from 0 to 59). The system keeps time on a 24-hour clock.

To enter the time of day, specify the time in the format described above. You should do this as soon as you bootstrap the system. The following example enters the time, 11:15:00 A.M.

```
.TIME 11:15
```

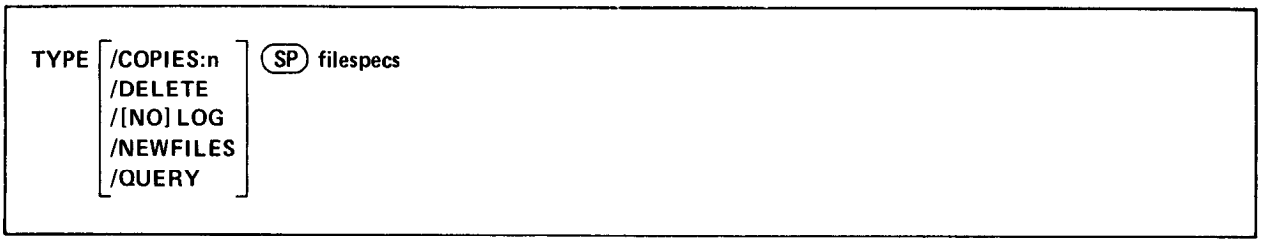
As this example shows, if you omit one of the arguments the system assumes 0. The system automatically resets the time each day at midnight.

To display the current time of day, type the TIME command without an argument, as this example shows.

```
.TIME
11:15:01
```

When the RT-11 system is installed, the clock rate is preset to 60 cycles. Consult the *RT-11 System Generation Manual* for information on setting the clock to a 50-cycle rate.

The TYPE command types (or prints) the contents of one or more files on the terminal.



In the command syntax illustrated above, filespecs represents the file or files to be typed. You can explicitly specify up to six files as input to the TYPE command. The system types the files in the order in which you specify them in the command line. You can also use wildcards in the file specification. In this case, the system types the files in the order in which they occur in the directory of the device you specify. If you specify more than one file, separate the files by commas. If you omit the file type for a file specification, the system assumes .LST. You can specify the entire command on one line, or you can rely on the system to prompt you for information. The TYPE command prompt is Files?.

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

/COPIES:n – Use this option to type more than one copy of the file. The meaningful range of values for the decimal argument, n, is from 2 to 32 (1 is the default). The following command, for example, types three copies of the file REPORT.LST on the terminal.

```
.TYPE/COPIES:3 REPORT
```

/DELETE – Use this option to delete a file after it is typed on the terminal. This option must appear following the command in the command line. The TYPE/DELETE operation does not ask you for confirmation before it executes. You must use /QUERY for this function. The following example types a BASIC program on the terminal, then deletes it from DX1:.

```
.TYPE/DELETE DX1:PROG1.BAS
```

/LOG – This option prints on the terminal the names of the files that were typed by the current command. Normally, the system prints a log only if there is a wildcard in the file specification. If you specify /QUERY, the query message replaces the log, unless you specifically type /LOG/QUERY in the command line. The following example shows a TYPE command and the resulting log.

```
.TYPE/LOG OUTFIL.LST
 Files copied:
DK:OUTFIL.LST    to TT:
```

/NOLOG – This option prevents a list of the files that were typed from printing on the terminal. You can use this option to suppress the log if you use a wildcard in the file specification.

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

```
.TYPE/NEWFILES *.LST
 Files copied:
DK:REPORT.LST    to TT:
```

/QUERY – *If you use this option, the system requests confirmation from you before it performs the operation.* /QUERY is particularly useful on operations that involve wildcards, when you may not be completely sure which files the system selected for an operation. Note that if you specify /QUERY in a TYPE command line that also contains a wildcard in the file specification, the confirmation messages that print on the terminal replace the log messages that would normally appear. You must respond to a query message by typing Y (or anything that begins with Y) and a carriage return to initiate execution of a particular operation. The system interprets any other response as NO and it does not perform the specific operation.

```
.TYPE/QUERY/DELETE *.LST
 Files copied/deleted:
DK:OUTFIL.LST    to TT:? NO
DK:REPORT.LST    to TT:? Y
```

The UNLOAD command makes handlers that were previously loaded non-resident, thus freeing the memory space they occupied.

UNLOAD (SP) **device[, . . . device]**

In the command syntax shown above, device represents the device handler to unload.

UNLOAD clears ownership for all units of the device type you specify. A request to unload the system device handler clears ownership for any assigned units for that device, but the handler itself remains resident. After you issue the UNLOAD command, the system returns any memory it frees to a free memory list. The background job eventually reclaims free memory. Note that if you interrupt an operation that involves magtapes or cassette, you must unload and then load (with the LOAD command) the appropriate device handler (MM, MT, or CT).

The system does not accept an UNLOAD command while a foreground job is running if the foreground job owns any units of that device. This is because a handler that the foreground job needs might become nonresident. You can unload a device while a foreground job is running if none of its units belong to the foreground job.

A special function of this command is to remove a terminated foreground job and reclaim memory, since the system does not automatically return the space occupied by the foreground job to the free memory list. The next command unloads the foreground job and frees the memory it occupied. This command is valid only if the foreground job is not running.

```
.UNLOAD RK:
```

The following command clears ownership of all units of RK if RK: is the system device.

```
.UNLOAD LP:,DT:
```

The next command releases the line printer and DECtape handlers and frees the area they previously held.

```
.UNLOAD FG
```

PART III
TEXT EDITING

You use an editor to create and modify textual material. PART III describes the RT-11 text editor, EDIT, and explains how to use it.

CHAPTER 5
TEXT EDITOR

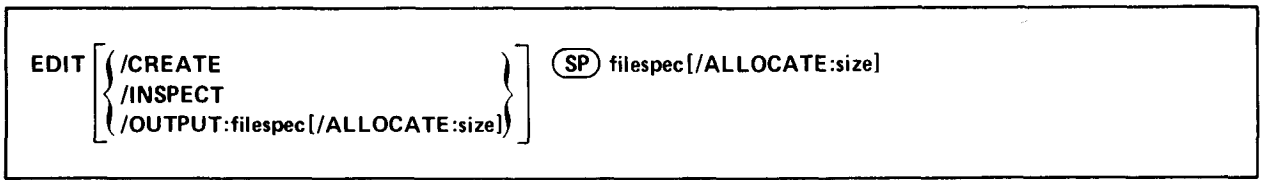
The text editor (EDIT) is a program that creates or modifies ASCII source files for use as input to other system programs such as the MACRO assembler or the FORTRAN compiler. EDIT, which accepts commands you type at the terminal, reads ASCII files from any input device, makes specific changes, and writes on any output device. EDIT allows efficient use of VT11 or VS60 display hardware, if they are part of the system configuration.

The editor considers a file to be divided into logical units called pages. A page of text is generally 50-60 lines long (delimited by form feed characters) and corresponds approximately to a physical page of a program listing. The editor reads one page of text at a time from the input file into its internal buffers where the page becomes available for editing. You can then use editing commands to:

- Locate text to be changed
- Execute and verify the changes
- List an edited page on the console terminal
- Output a page of text to the output file.

5.1 CALLING AND USING EDIT

You can call the text editor when you are at monitor level. The syntax of the command is:



See Section 4.4 for a description of the EDIT command and its options.

5.2 MODES OF OPERATION

Normally, the editor operates in either command mode or text mode. In command mode the editor interprets all input you type on the keyboard as commands to perform some operation. In text mode the editor interprets all typed input as text to replace, insert into, or append to the contents of the text buffer.

Immediately after being loaded into memory and started, the editor is in command mode. EDIT prints an asterisk at the left margin of the console terminal page to indicate that it is ready to accept a command. Terminate all commands by pressing the ESCAPE key twice in succession. Execution of commands proceeds from left to right. Should EDIT encounter an error before it begins execution of a command string, it prints an error message followed by an asterisk at the beginning of a new line, indicating that it is still in command mode and awaiting a legal command. EDIT does not execute the command in error or any succeeding command. You should retype the command correctly.

To enter text mode, type a command that must be followed by a text string. These commands insert, replace, exchange, or otherwise manipulate text. When you type one of these commands, EDIT recognizes all succeeding characters as part of the text string until it encounters an ESCAPE character. The ESCAPE terminates the text string and causes the editor to reenter command mode.

You can use a special editing mode, called immediate mode, whenever the VT-11 display hardware is running. Section 5.7.2 describes this mode.

5.3 SPECIAL KEY COMMANDS

Table 5-1 lists the EDIT key commands. Type a control command by holding down the CTRL key while typing the appropriate character.

Table 5-1 EDIT Key Commands

Key	Explanation
ESCAPE, ALTMODE, or SEL	Echoes $. A single ESCAPE terminates a text string. A double ESCAPE (two consecutive ESCAPEs) executes the command string. For example: `*GMOV A,B$-1D$$` The first ESCAPE ($) terminates the text object (MOV A,B) of the Get command. The double ESCAPE ($$) terminates the Delete command and executes the entire command string. In this example, the character B will be deleted as a result of execution.

CTRL/C	Echoes at the terminal as ^C. If EDIT encounters a CTRL/C as a command in command mode, it terminates execution and returns control to the monitor. You can restart the editor by typing R EDIT or REENTER in response to the monitor's prompt. If EDIT encounters a CTRL/C in a text object, EDIT includes the CTRL/C in the text object, just like any other character. If the editor is executing a lengthy command and you want to stop EDIT, type two CTRL/C commands in succession. This will abort the command, generate the ?EDIT-F-COMMAND ABORTED error message, and return the editor to command mode. For example:
	
	`*I^C^C^C$$`
	`*^C$$`
	
	In the first command, the three CTRL/C characters are part of the text object of the Insert command. EDIT treats them like any other character. In the second command string, the CTRL/C occurs at command level, and causes the editor to terminate.
	
	If no commands (other than CLOSE) are executed between the time you terminate the editor and the time you issue a REENTER command, the text buffer is preserved exactly as it was at program termination. However, only the text buffer is preserved. The input and output files are closed, and the save and macro buffers are reinitialized.
	
	If you inadvertently terminate an editing session before the output file can be closed, you can often use the monitor CLOSE command to make permanent the portion of the output file that has already been written (see Section 4.4). You can then reenter the editor, open a new output file, and continue the editing session.

(Continued on next page)

Table 5-1 (Cont.) EDIT Key Commands

Key	Explanation
CTRL/O	Echoes ^O and a carriage return. Inhibits printing on the terminal until completion of the current command string. Typing a second CTRL/O resumes output.
CTRL/U	Echoes ^U and a carriage return. Deletes all the characters on the current terminal input line. (Equivalent to pressing the RUBOUT key until all the characters back to the beginning of the line are deleted.)
RUBOUT or DELETE	Deletes a character from the current command line; echoes a backslash followed by the character deleted. Each succeeding RUBOUT you type deletes and echoes another character. An enclosing backslash prints when you type a key other than RUBOUT. This erasure is done from right to left. Since EDIT accepts multiple line commands, RUBOUT can delete past the carriage return/line feed combination and delete characters on the previous line. You can use RUBOUT in both command and text modes.
TAB	Spaces to the next tab stop. Tab stops are positioned every eight spaces on the terminal; pressing the TAB key causes the carriage to advance to the next tab position.
CTRL/X	Echoes ^X and a carriage return. CTRL/X causes the editor to ignore the entire command string you are currently entering. The editor prints a carriage return/line feed combination and an asterisk to indicate that you can enter another command. For example: *IABCD EFGH^X * A CTRL/U would cause only deletion of EFGH; CTRL/X erases the entire command.

5.4 COMMAND STRUCTURE

EDIT commands fall into eight general categories. Table 5-2 lists these categories and the commands they include.

Table 5-2 EDIT Command Categories

Category	Commands	Section
File open and close	Edit Backup	5.6.1.3
	Edit Read	5.6.1.1
	Edit Write	5.6.1.2
	End File	5.6.1.4
File input/output	EXit	5.6.2.4
	Next	5.6.2.3
	Read	5.6.2.1
	Write	5.6.2.2

(Continued on next page)

Table 5-2 (Cont.) EDIT Command Categories

Category	Commands	Section
Immediate mode	ESCAPE	5.7.2
	CTRL D	5.7.2
	CTRL G	5.7.2
	CTRL N	5.7.2
	CTRL V	5.7.2
	RUBOUT	5.7.2
Pointer location	Advance	5.6.3.3
	Beginning	5.6.3.1
	Jump	5.6.3.2
Search	Find	5.6.4.2
	Get	5.6.4.1
	Position	5.6.4.3
Text listing	List	5.6.5.1
	Verify	5.6.5.2
Text modification	Change	5.6.6.4
	Delete	5.6.6.2
	eXchange	5.6.6.5
	Insert	5.6.6.1
	Kill	5.6.6.3
Utility	Edit Console	5.7.1
	Edit Display	5.7.1
	Edit Lower	5.6.7.6
	Edit Upper	5.6.7.6
	Edit Version	5.6.7.5
	Executive Macro	5.6.7.4
	Macro	5.6.7.3
	Save	5.6.7.1
	Unsave	5.6.7.2

The general syntax for all the EDIT commands, with the exception of the immediate mode commands, is:

[n]C[text]$

or

[n]C$

where

n represents one of the legal arguments from Table 5-3.

C represents a 1- or 2-letter command.

text represents a string of successive ASCII characters.

As a rule, commands are separated from one another by a single ESCAPE; however, if the command requires no text, the separating ESCAPE is not necessary. Commands are terminated by a single ESCAPE; typing a second ESCAPE begins execution. (You use ESCAPE differently when immediate mode is in effect; Section 5.7.2 details its use in this case.)

The syntax of display editor commands is somewhat different from the normal editing command format, and is described in Section 5.7.

5.4.1 Arguments

An argument is positioned before a command letter. It specifies either the particular portion of text to be affected by the command or the number of times to perform the command. With some commands, this specification is implicit and no argument is needed; other editing commands require an argument. Table 5-3 lists the possible arguments and their meanings.

Table 5-3 Command Arguments

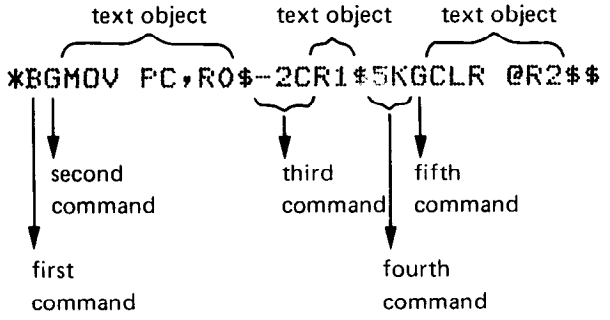
Argument	Meaning
n	Stands for any integer in the range -16383 to +16383 and may, except where noted, be preceded by a plus (+) or minus (-) sign. If no sign precedes n, it is assumed to be a positive number. The absence of n implies a 1 (or -1 if a minus sign precedes a command). n can represent the number of characters or lines forward or backward (+ or -) to move the pointer, or it can represent the number of times to execute the operation.
0	Indicates the text between the beginning of the current line and the reference pointer (see Section 5.4.3).
/	Refers to the text between the reference pointer and the end of the text in the buffer.
=	Use only with the J, D, and C commands to represent -n, where n is equal to the length of the last text argument used.

The roles of all arguments are explained more specifically in the following sections.

5.4.2 Command Strings

All EDIT command strings are terminated by two successive ESCAPE characters. Use spaces, carriage returns, and line feeds within a command string to increase command readability. EDIT ignores them unless they appear in a text string. Commands to insert text can contain text strings that are several lines long. Each line you enter is terminated by the carriage return key, which inserts both a carriage return and a line feed character into the text. The entire command is terminated by a double ESCAPE.

You can string several commands together and execute them in sequence. For example:



where

B	is the first command.
GMOV PC,R0	is the second command (MOV PC,R0 is the text object).
-2CR1	is the third command (R1 is the text object).
5K	is the fourth command.
GCLR @R2	is the fifth command (CLR @R2 is the text object).
$	separates the end of each text object from the following command.
$$	executes the commands.

Execution of a command string begins when you type the double ESCAPE and proceeds from left to right. Except when they are part of a text string, EDIT ignores spaces, carriage returns, line feeds, and single ESCAPEs. For example:

```
*BGMOV R0$=CCLR R1$AV$$
```

You can also type this command as:

```
*B$ GMOV R0$
=CCLR R1$
A$ V$$
```

Execution of the two commands will be the same.

5.4.3 The Current Location Pointer

Most EDIT commands function with respect to a movable reference pointer that is normally located between the most recent character operated upon and the next character in the buffer. It is important to think of this pointer as being between two characters and never directly on a character. At the start of editing operations, the pointer precedes the first character in the buffer, although it is not displayed on the console terminal. At any given time during the editing procedure, think of the pointer as representing the current position of the editor in the text. The pointer moves during editing operations according to the type of editing operation being performed. Refer to text in the buffer as so many characters or lines preceding or following the pointer.

5.4.4 Character- and Line-Oriented Command Properties

Edit commands are either character-oriented or line-oriented: character-oriented commands affect a specified number of characters preceding or following the pointer; line-oriented commands operate on entire lines of text.

The argument of character-oriented commands specifies the number of characters in the buffer on which to operate. If n is unsigned (positive), the command operates in a forward direction. If n is preceded by a minus sign (negative), the command moves the reference pointer backwards. (LF), (RET), and null characters, although not printed. are embedded in text lines, counted as characters in character-oriented commands, and treated as any other text characters. When you press the (RET) key, both a carriage return and a line feed character are inserted into the text. For example, assume the pointer is positioned as indicated in the following text (↑ represents the current position of the pointer):

```
MOV   #VECT,R2(RET)(LF)
                      ↑
CLR   @R2(RET)(LF)
```

The EDIT command -2J moves the pointer back two characters to precede the carriage return character.

```
MOV    #VECT, R2↑(RET)(LF)
CLR    @R2 (RET)(LF)
```

The command 10J advances the pointer forward by ten characters and places it between the (RET) and (LF) characters at the end of the second line. Note that the tab character preceding @R2 is also counted as a single character.

```
MOV    #VECT,R2 (RET)(LF)
CLR    @R2 (RET)↑(LF)
```

Finally, to place the pointer after the C in the first line, use a -14J command. The J (Jump) command is explained in Section 5.6.3.2.

```
MOV    #VECT↑,R2(RET) (LF)
CLR    @R2 ↑(RET) (LF)
```

When you use line-oriented commands, the argument of the commands specifies the number of lines on which to operate. Because EDIT counts the line-terminating characters to determine the number of lines on which to operate, an argument, n, does not affect the same number of lines forward (positive) as it affects backward (negative). For example, the argument -1 applies to the line beginning with the first character following the second previous end-of-line and ending with the character preceding the pointer. The argument 1 in a line-oriented command, however, applies to the text beginning with the first character following the pointer and ending at the first end-of-line. Thus, if the pointer is at the center of the line, the argument -1 affects one and one-half lines backwards from the pointer and the argument 1 affects one-half line beyond the pointer.

For example, assume the buffer contains:

```
MOV    ↑PC,R1 (RET)(LF)
ADD    ↑#DRIV-.,R1(RET)(LF)
MOV    #VECT,R2(RET)(LF)
CLR    @R2(RET)(LF)
```

The command to advance the pointer one line (1A) causes the following change:

```
MOV    PC,R1(RET)(LF)
↑ADD   #DRIV-.,R1(RET)(LF)
↑MOV   #VECT,R2(RET)(LF)
CLR    @R2(RET)(LF)
```

The command 2A moves the pointer over two (RET)(LF) combinations to precede the fourth line:

```
MOV    PC,R1(RET)(LF)
ADD    #DRIV-.,R1(RET)(LF)
MOV    #VECT,R2(RET)(LF)
↑CLR   @R2 (RET)(LF)
```

Assume the buffer contains:

```
MOV    PC,R1(RET)(LF)
ADD    #DRIV-.,R1(RET)(LF)
MOV    #VECT↑,R2(RET)(LF)
CLR    @R2 ↑(RET)(LF)
```

A command of -1A moves the pointer back by one and one-half lines to precede the second line.

```
MOV   PC,R1(RET)(LF)
↑ADD  #DRIV-.,R1(RET)(LF)
MOV   #VECT,R2(RET)(LF)
CLR   @R2(RET)(LF)
```

Now a command of -1A moves the pointer back by only one line.

```
↑MOV  PC,R1(RET)(LF)
ADD   #DRIV-.,R1(RET)(LF)
MOV   #VECT,R2(RET)(LF)
CLR   @R2(RET)(LF)
```

5.4.5 Command Repetition

You can execute portions of a command string more than once by enclosing the portion in angle brackets (<>) and preceding the left angle bracket with the number of iterations you desire. The syntax is:

n<command>

For example:

C1$C2$n<C3$C4$>C5$$

where

C represents a command.

n represents an iteration argument.

Commands C1 and C2 each execute once, then commands C3 and C4 execute n times. Finally, command C5 executes once and the command line is finished. The iteration argument (n) must be a positive number (in the range 1 through 16,383) and, if you do not specify it, it is assumed to be 1. If the number is negative or too large, an error message prints. You can nest iteration brackets up to 20 levels. EDIT checks command lines to make certain the brackets are correctly used and match prior to execution.

Essentially, enclosing a portion of a command string in iteration brackets and preceding it with an iteration argument (n) is equivalent to typing that portion of the string n times. For example:

```
*BGAAA$3<-DIB$-J>V$$
*BGAAA$-DIB$-J-DIB$-J-DIB$-JV$$
```

These two strings are equivalent.

Similarly, the following two strings are equivalent:

```
*B3<2<AD>V>$$
*BADADVADADVADADV$$
```

The following bracket structures are examples of legal usage:

```
<<><<<><>>>>
<<<>>><><>
```

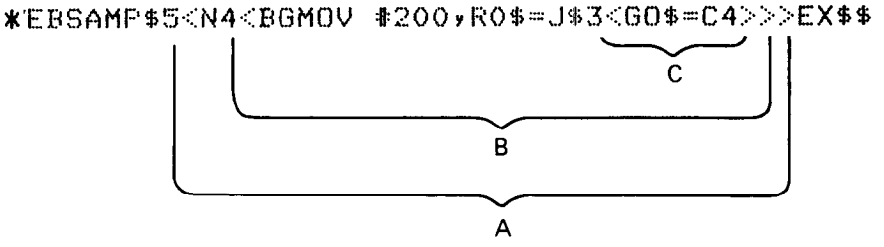
The following bracket structures are examples of illegal combinations that will cause an error message since the brackets are not properly matched:

```
><><
<<<>>
```

During command repetition, execution proceeds from left to right until a right bracket is encountered. EDIT then returns to the last left bracket encountered, decreases the iteration counter, and executes the commands within the brackets. When the counter is decreased to 0, EDIT looks for the next iteration count to the left and repeats the same procedures. The overall effect is that EDIT works its way to the innermost brackets and then works its way back again. The most common use for iteration brackets is found in commands, such as Unsave (U), that do not accept repeat counts. For example:

```
*3<U>$$
```

Assume you want to read a file called SAMP (stored on device DK:), and you want to change the first four occurrences of the instruction MOV #200,R0 on each of the first five pages to MOV #244,R4. Enter the following command line:

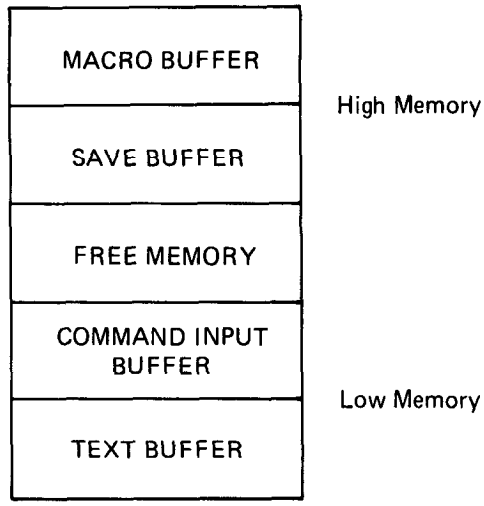


The command line contains three sets of iteration loops (A,B,C) and executes as follows:

Execution initially proceeds from left to right; EDIT opens the file SAMP for input and reads the first page into memory. EDIT moves the pointer to the beginning of the buffer and initiates a search for the character string MOV #200,R0. When it finds the string, EDIT positions the pointer at the end of the string, but the =J command moves the pointer back, so that it is positioned immediately preceding the string. At this point, execution has passed through each of the first two sets of iteration loops (A,B) once. The innermost loop (C) is next executed three times, changing the 0s to 4s. Control now moves back to pick up the second iteration of loop B, and again moves from left to right. When loop C has executed three times, control again moves back to loop B. When loop B has executed a total of four times, control moves back to the second iteration of loop A, and so forth, until all iterations have been satisfied.

5.5 MEMORY USAGE

The memory area used by the editor is divided into four logical buffers as follows:



The text buffer contains the current page of text you are editing, and the command input buffer holds the command you are currently typing at the terminal. If a command you are currently entering is within ten characters of exceeding the space available in the command buffer, the following message prints on the terminal.

```
?EDIT-W-Command buffer almost full
```

If you can complete the command within ten characters, you can finish entering the command; otherwise you should press the ESCAPE key twice to execute that portion of the command line already completed. The message prints each time you enter a character in one of the last ten spaces.

If you attempt to enter more than ten characters, EDIT prints the following message and aborts the command.

```
?EDIT-F-Command buffer full¦no command(s) executed
```

This will never occur if you heed the preceding warning and terminate the command immediately.

The save buffer contains text stored with the Save (S) command, and the macro buffer contains the command string macro entered with the Macro (M) command. (Both commands are explained in Section 5.6.7.)

EDIT does not allocate space for the macro and save buffers until an M or S command executes. Once you enter an M or S command, a 0M or 0U (Unsave) command returns that space to the free area.

The size of each buffer automatically expands and contracts to accommodate the text you are entering; if there is not enough space available to accommodate required expansion of any of the buffers, EDIT prints the error message:

```
?EDIT-F-Insufficient memory
```

5.6 EDITING COMMANDS

This section describes the commands and procedures required to:

- Read text from the input files to the buffer
- Create a backup version of the file
- List the contents of the buffer on the terminal
- Move the reference pointer

- Locate specific characters or strings of characters within the text buffer
- Insert, relocate, or delete text in the buffer
- Close the output file
- Terminate the editing session.

The following sections are arranged, in order, by category of command function, as illustrated in Table 5-2.

5.6.1 File Open and Close Commands

You can use file open and close commands to:

- Open an existing file for input and prepare it for editing
- Open a file for output of newly created or edited text
- Open an existing file for editing and create a backup version of it
- Close an open output file.

5.6.1.1 Edit Read – The Edit Read (ER) command opens an existing file for input and prepares it for editing. Only one file can be open for input at a time.

The syntax of the command is:

```
ERdev:filnam.typ$
```

The string argument (dev:filnam.typ) is limited to 19 characters and specifies the file to be opened. If you do not specify a device, DK: is assumed. If a file is currently open for input, EDIT closes the file and opens the new one.

Edit Read does not input a page of text nor does it affect the contents of the other user buffers.

You can use Edit Read on a file that is already open to close that file for input and reposition EDIT at its beginning. The first Read command following any Edit Read command inputs the first page of the file.

```
*ERDT1:SAMP.MAC$$
```

This command string, for example, opens the file SAMP.MAC on device DT1: for input.

NOTE

If you enter EDIT with the monitor EDIT/INSPECT or EDIT/OUTPUT command, an Edit Read command is automatically performed on the file named in the EDIT command.

5.6.1.2 Edit Write – The Edit Write (EW) command opens a file for output of newly created or edited text. However, no text is output and the contents of the buffers are not affected. Only one file can be open for output at a time. EDIT closes any output files currently open and preserves any edits made to the file.

The syntax of the command is:

```
EWdev:filnam.typ[n]$
```

The string argument (dev:filnam.typ[n]) is limited to 19 characters and is the name you assign to the output file being opened. If you do not specify a device, DK: is assumed. [n] is an optional decimal number that represents the length of the file to be opened. Note that the square brackets [] are part of the argument, n. You must type them. If you do not specify [n], the default size will be used. That is, the system will choose the larger of 1) one-half the largest available space, and 2) the second largest available space. If this is not adequate for the output file size, you must close this file and open another when this one becomes full. You should use the [n] construction whenever there is doubt as to whether enough space is available on the device for one output file.

If a file with the same name already exists on the device, EDIT deletes the existing file when you type an Exit, End File, or another Edit Write command. EDIT prints the warning message:

```
?EDIT-W-Superseding existing file
```

The following command, for example, opens for output the file FILE.BAS on device DK: and allocates 11 blocks of space for it.

```
*EWFILE.BAS[11]$$
```

NOTE

If you enter EDIT with the monitor EDIT/CREATE command, an Edit Write command is automatically performed on the file named in the EDIT command. If you enter EDIT with the monitor EDIT/OUTPUT command, an Edit Write is automatically performed on the file named with the /OUTPUT option.

5.6.1.3 Edit Backup – Use the Edit Backup (EB) command to open an existing file for editing and at the same time create a backup version of the file. EDIT closes any input and output file currently opened. No text is read or written with this command.

The syntax of the command is:

```
EBdev:filnam.typ[n]$
```

The device designation, file name, and file type are limited to 19 characters. If you do not specify a device, DK: is assumed. [n] is optional and represents the length of the file to be opened; if you do not specify [n], the default size will be used. That is, the system will choose the larger of 1) one-half the largest available space, and 2) the second largest available space.

The file you indicate in the command line must already exist on the device you designate, since text will be read from this file as input. At the same time, EDIT opens an output file under the same file name and file type. When the output file is closed, EDIT renames the original file (used as input) with the current file name and a .BAK file type and deletes any previous file with this file name and a .BAK file type. EDIT closes the new output file and assigns it the name you specify in the EB command. This renaming of files takes place when an Exit, End File, or subsequent Edit Write or Edit Backup command executes. If you terminate the editing session with a CTRL/C command before the output file is closed, the new output file is not made permanent, and the renaming of the current version to .BAK does not take place.

```
*EBSY:BAS1.MAC$$
```

[illegible] deletes any previous version of BAS1.BAK.

NOTE

In EB, ER, and EW commands, leading spaces between the command and the file name are not permitted because EDIT assumes the file name to be a text string. All dev:filnam.typ specifications for EB, ER, and EW commands conform to the RT-11 conventions for file naming and are identical to file names entered in command strings used with other system programs.

If you enter EDIT with an unqualified monitor EDIT command, an Edit Backup command is automatically performed on the file named in the EDIT command.

5.6.1.4 End File – The End File (EF) command closes the current output file and makes it permanent. You can use the EF command to create an output file from a section of a large input file or to close an output file that is full before you open another file. Modifiers are illegal with an EF command. Note that an implied EF command is included in EW and EB commands.

The syntax of the command is:

EF

Table 5-4 illustrates the relationship between the file open and close commands and the buffers and files themselves.

Table 5-4 EDIT Commands and File Status

Command	Input File	Text Buffer	Output File
ERXXX$	Opens XXX for input; closes existing input file, if any	Unchanged	Unchanged
EWXXX$	Unchanged	Unchanged	Opens XXX for output; closes existing output file, if any; performs .BAK renaming if EB is in effect
EBXXX$	Opens XXX for input; closes existing input file, if any	Unchanged	Opens a temporary file for output; closes existing output file, if any; performs .BAK renaming if EB is in effect
EF$	Unchanged	Unchanged	Closes output file; performs .BAK renaming if EB is in effect
EX$	Copies to output file	Copies to output file	Closes output file after copying complete; performs .BAK renaming if EB is in effect

5.6.2 File Input/Output Commands

You use file input/output commands to:

- Read text from an input file into the buffer
- Copy lines of text from the buffer into an output file
- Terminate the editing session.

5.6.2.1 Read – Before you can edit text, you must read the input file into the buffer. The Read (R) command reads a page of text from the input file (previously specified in an ER or EB command) and appends it to the current contents, if any, of the text buffer.

The command is:

```
R
```

No arguments are used with the R command. If text resides in the buffer prior to the R command, the pointer does not move; however, if no text resides in the buffer, the pointer is placed at the beginning of the buffer. EDIT transfers text to the buffer until one of the following conditions occurs:

1. A form feed character, signifying the end of the page, is encountered.
2. The text buffer is 500 characters from being full. (When this condition occurs, the Read command inputs up to the next carriage return/line feed combination, then returns to command mode. An asterisk prints as though the read were complete, but text will not have been fully input).
3. An end-of-file is encountered, (the ?EDIT-F-END OF INPUT FILE message prints when all text in the file has been read into memory and no more input is available).

The maximum number of characters that you can bring into memory with an R command depends on the system configuration and the memory requirements of other system components. EDIT prints an error message if the read exceeds the memory available or if no input is available.

The following example edits a file using the EB and R commands.

```
*EBSJK1.BAS$$
```

This command opens SJK1.BAS on DK: and permits modification.

```
*R/L$$
THIS IS PAGE ONE OF
FILE SJK1.BAS.
```

This command reads the first page of SJK1.BAS into the buffer. The pointer is placed at the beginning of the buffer. /L lists the contents of the buffer on the terminal beginning at the pointer and ending with the last character in the buffer.

5.6.2.2 Write – The Write (nW) command copies lines of text from the text buffer to the output file (as specified in the EW or EB command). The contents of the buffer are not altered and the pointer is left unchanged (unless an output error occurs).

NOTE

EDIT uses a system of intermediate buffers to store output before it actually writes the data to an output file. The Write command logically writes to the file, but actual output to a

device does not occur until the intermediate buffer fills. When the editor closes a file (that is, after you issue an EF, EB, EX, or EW command), the editor writes from the buffer to the file and the file is complete. If the editor does not close a file (if you exit with CTRL/C and use the CLOSE command), it is possible that the output file will be missing the last 512 characters.

The syntax of the command is:

nW

The argument you supply with the W command determines the lines of text to copy. Table 5-5 lists the arguments for the W command and their effect.

Table 5-5 Write Command Arguments

Argument	Meaning
n	Writes n lines of text beginning at the pointer and ending with the nth end-of-line character to the output file.
-n	Writes n lines of text to the output file beginning with the first character on the -nth line and terminating at the pointer.
0	Writes to the output file the current line up to the pointer.
/	Writes to the output file the text between the pointer and the end of the buffer.

If the buffer is empty when the write executes, no characters are output.

The following examples illustrate the use of the W command.

```
*5W$$
```

This command writes the five lines of text following the pointer into the current output file.

```
*-2W$$
```

This command writes the two lines of text preceding the pointer into the current output file.

```
*B/W$$
```

This command writes the entire text buffer to the current output file.

NOTE

If an output file fills while a Write command is executing, EDIT prints the ?EDIT-F-OUTPUT FILE FULL message. In this case, EDIT positions the reference pointer after the last character it wrote successfully. You can then use the following recovery procedure:

1. Close the current output file. (EF command)
2. Open a new output file. (EW command)
3. Delete the characters just written by using -nD or -nK, where n is any arbitrary number that exceeds the number of lines or characters in the buffer.
4. Resume output.

5.6.2.3 Next – The Next (nN) command writes the contents of the text buffer to the output file, deletes the text from the buffer, and reads the next page of the input file into the buffer. The pointer is positioned at the beginning of the buffer. The syntax of the command is:

```
nN
```

If you specify the argument n with the Next command, the sequence is executed n times.

If EDIT encounters the end of the input file when trying to execute an N command, it prints ?EDIT-F-END OF INPUT FILE to indicate that no further text remains in the input file. Since the contents of the buffer has already been transferred to the output file, the buffer is empty.

Using the N command is a quick way to write edited text to the output file and set up the next page of text in the buffer. The N command functions as though it were a combination of the Write, Delete, Read, and Beginning commands. (Delete is a text modification command, described in Section 5.6.6.2; the Beginning command is a pointer relocation command, described in Section 5.6.3.1.) Using the N command with an argument is a convenient way to set up text in the buffer, if you already know its page location. The N command operates in a forward direction only; therefore, you cannot specify negative arguments with an N command.

In the following example, an N command copies an input file with more than one page of text to the output file.

```
*EBDK:TEST.MAC$$
```

This command opens the file TEST.MAC on device DK: and creates a new file entitled TEST.MAC for output.

```
*N/L$$
THIS IS PAGE ONE OF
FILE TEST.MAC.
```

This command reads the first page of the input file, TEST.MAC, into the buffer and lists the entire page on the terminal.

```
*N/L$$
?EDIT-F-End of input file
*
```

This command transfers the contents of the buffer to the output file, clears the buffer, and encounters the end of the file. Because it cannot complete the N sequence, EDIT prints ?EDIT-F-END OF INPUT FILE on the terminal. The buffer is empty and the entire input file has been written to the output file.

5.6.2.4 EXit – Type the Exit (EX) command to terminate an editing session. The Exit command does the following:

- Writes the text buffer to the output file
- Transfers the remainder of the input file to the output file
- Closes all open files
- Renames the backup file with a .BAK file type if an EB command is in effect
- Returns control to the monitor.

The command is:

EX

No arguments are accepted. Essentially, Exit copies the remainder of the input file into the output file and returns to the monitor. Exit is legal only when there is an output file open. If an output file is not open and you want to terminate the editing session, return to the monitor with CTRL/C.

NOTE

You must issue an EF or EX command in order to make an output file permanent. If you use CTRL/C to return to the monitor without issuing an EF command, the current output file will not be saved. (You can, however, make permanent that portion of the text file that has already been written out by using the monitor CLOSE command.)

An example of the contrasting uses of the EF and EX commands follows. Assume an input file, SAMPLE, contains several pages of text. The first and second pages of the file will be made into separate files called SAM1 and SAM2, respectively; the remaining pages of text will then make up the file SAMPLE. This can be done using these commands:

```
*EWSAM1$$
*ERSAMPLE$$
*RNEF$$
*EWSAM2$$
*NEF$$
*EWSAMPLE$EX$$
```

Note that the EF commands are not strictly necessary in this example since the EW command closes a currently open output file before opening another.

5.6.3 Pointer Relocation Commands

Pointer relocation commands allow you to change the current location of the reference pointer within the text buffer.

5.6.3.1 Beginning – The Beginning (B) command moves the current location of the pointer to the beginning of the text buffer.

The command is:

B

There are no arguments.

For example, assume the buffer contains:

```
MOVB  5(R1),@R2
ADD   R1,(R2)+
CLR   @R2
MOVB  6(R1),@R2
```

The B command moves the pointer to the beginning of the text buffer.

```
*B$$
```

The text buffer now looks like this:

```
↑MOVB  5(R1),@R2
 ADD   R1,(R2)+
 CLR   @R2
 MOVB  6(R1),@R2
```

5.6.3.2 Jump – The Jump (nJ) command moves the pointer past the specified number of characters in the text buffer. The syntax of the command is:

nJ

Table 5-6 shows the arguments for the J command and their meanings.

Table 5-6 Jump Command Arguments

Argument	Meaning
(+ or -) n	Moves the pointer (forward or backward) n characters.
0	Moves the pointer to the beginning of the current line (equivalent to 0A).
/	Moves the pointer to the end of the text buffer (equivalent to /A).
=	Moves the pointer backward n characters, where n equals the length of the last text argument used.

Negative arguments move the pointer toward the beginning of the buffer; positive arguments move it toward the end. Jump treats carriage returns, line feeds, and form feed characters the same as any other character, counting one buffer position for each one.

The following commands illustrate the use of the J command.

```
*3J$$
```

This command moves the pointer ahead three characters.

```
*-4J$$
```

This command moves the pointer back four characters.

```
*B$$GABC$$=J$$
```

This command moves the pointer so that it immediately precedes the first occurrence of ABC in the buffer.

5.6.3.3 Advance – The Advance (nA) command is similar to the Jump command except that it moves the pointer a specific number of lines (rather than single characters) and leaves it positioned at the beginning of the line. The syntax of the command is:

nA

Table 5-7 lists the arguments for the A command and their meanings.

Table 5-7 Advance Command Arguments

Argument	Meaning
n	Moves the pointer forward n lines and positions it at the beginning of the nth line.
-n	Moves the pointer backward past n carriage return/line feed combinations and positions it at the beginning of the -nth line.
0	Moves the pointer to the beginning of the current line (equivalent to 0J).
/	Moves the pointer to the end of the text buffer (equivalent to /J).

Following are examples that use the A command.

```
*3A$$
```

This command moves the pointer ahead three lines.

Assume the buffer contains:

```
CLR     @R2
        ↑
```

The following command moves the pointer to the beginning of the current line:

```
*0A$$
```

Now the buffer looks like this:

```
↑CLR    @R2
```

5.6.4 Search Commands

Use search commands to locate specific characters or strings of characters within the text buffer.

NOTE

Search commands always have positive arguments. They search ahead in the file. This means that you cannot search for a character string that has already been written to the output file. To do this, you must first close the currently open files (with EX) then edit the file that was just used for output (with EB).

5.6.4.1 Get – The Get (nG) command is the basic search command in EDIT. It searches the current text buffer for the nth occurrence of a specific text string starting at the current location of the pointer. If you do not supply the argument n, EDIT searches for the first occurrence of the text object. The search terminates when EDIT either finds the nth occurrence or encounters the end of the buffer. If the search is successful, EDIT positions the pointer to follow the last character of the text object. EDIT notifies you of an unsuccessful search by printing ?EDIT-F-SEARCH FAILED. In this instance, EDIT positions the pointer after the last character in the buffer.

The syntax of the command is:

```
nGtext$
```

The argument (n) must be positive. If you omit it, EDIT assumes it to be 1.

The text string can be any length and must immediately follow the G command. EDIT makes the search on the portion of the text between the pointer and the end of the buffer.

For example, assume the pointer is at the beginning of the buffer shown below.

```
↑MOV    PC,R1
 ADD    #DRIV-.,R1
 MOV    #VECT,R2
 CLR    @R2
 MOVB   5(R1),@R2
 ADD    R1,(R2)+
 CLR    @R2
 MOVB   6(R1),@R2
```

The following command searches for the first occurrence of the characters ADD following the pointer and places the pointer after the searched characters.

```
*GADD$$
```

Now the buffer looks like this:

```
MOV    PC,R1
ADD↑   #DRIV-.,R1
```

The next command searches for the third occurrence of the characters @R2 following the pointer and leaves the pointer immediately following the text object.

```
*3G@R2$$
```

The buffer is changed to:

```
ADD    R1,(R2)+
CLR    @R2↑
```

After successfully completing a search command, EDIT positions the pointer immediately following the text object. Using a search command in combination with =J places the pointer in front of the text object, as follows:

```
*GTEST$=J$$
```

This command combination places the pointer before TEST in the text buffer.

5.6.4.2 Find – The Find (nF) command starts at the current pointer location and searches the entire input file for the nth occurrence of the text string. If EDIT does not find the nth occurrence of the text string in the current buffer, it automatically performs a Next command and continues the search on the new text in the buffer. When the search is successful, EDIT leaves the pointer immediately following the nth occurrence of the text string. If the search fails (i.e., EDIT detects the end-of-file for the input file and does not find the nth occurrence of the text string), EDIT prints ?EDIT-F-SEARCH FAILED. In this instance, EDIT positions the pointer at the beginning of an empty text buffer. When you use the F command, EDIT deletes the contents of the buffer after writing it to the output file.

```
nFtext$
```

The argument (n) must be positive. EDIT assumes it to be 1 if you do not supply another value.

You can use an F command to copy all remaining text from the input file to the output file by specifying a non-existent text object. The Find command functions like a combination of the Get and Next commands.

The following example uses the F command.

```
*2FMOVB     6(R1),@R2$$
```

This command searches the entire input file for the second occurrence of the text string MOVB 6(R1),@R2. EDIT places the pointer following the text string. EDIT writes the contents of each unsuccessfully searched buffer to the output file.

5.6.4.3 Position – The Position (nP) command is identical to the find (F) command with one exception. The F command transfers the contents of the text buffer to the output file as each page is unsuccessfully searched, but the P command deletes the contents of the buffer after it is searched, without writing any text to the output file.

The syntax of the command is:

```
nPtext$
```

The argument (n) must be positive. If you omit it, EDIT assumes it to be 1.

The nP command searches each page of the input file for the nth occurrence of the text object starting at the pointer and ending with the last character in the buffer. If EDIT finds the nth occurrence, it positions the pointer following the text object, deletes all pages preceding the one containing the text object, and positions the page containing the text object in the buffer.

If the search is unsuccessful, EDIT clears the buffer and does not transfer any text to the output file. EDIT positions the pointer at the beginning of an empty text buffer.

The position command is a combination of the Get, Delete, and Read commands; it is most useful as a means of placing the pointer in the input file. For example, if your aim in the editing session is to create a new file from the second half of the input file, a position search saves time.

The following example uses the P command.

```
*P3$$
```

This command searches the input file for the first occurrence of the text object, 3. EDIT positions the pointer after the text object.

```
*0L$$
INPUT FILE PAGE 3
```

The command lists on the terminal the current line up to the pointer.

5.6.5 Text Listing Commands

5.6.5.1 List – The List (nL) command prints at the terminal lines of text as they appear in the buffer. The syntax of the command is:

```
nL
```

An argument preceding the L command indicates the portion of text to print. For example, the command, 2L, prints on the terminal the text beginning at the pointer and ending with the second end-of-line character. The pointer is not altered by the L command. Table 5-8 lists arguments and their effect upon the list command.

Table 5-8 List Command Arguments

Argument	Meaning
n	Prints at the terminal n lines beginning at the pointer and ending with the nth end-of-line character.
-n	Prints all characters beginning with the first character on the -nth line and terminating at the pointer.
0	Prints the current line up to the pointer. Use this command to locate the pointer within a line.
/	Prints the text between the pointer and the end of the buffer.

These examples illustrate the use of the L command.

```
*-2L$$
```

This command prints all characters starting at the second preceding line and ending at the pointer.

```
*4L$$
```

This line prints all characters beginning at the pointer and terminating at the 4th carriage return/line feed combination.

Assuming the pointer location is:

```
MOVB  5(R1),@R2
ADD↑  R1,(R2)+
```

The following command prints the previous one and one-half lines up to the pointer:

```
*-1L$$
```

The terminal output looks like this:

```
MOVB  5(R1),@R2
ADD
```

5.6.5.2 Verify – The Verify (V) command prints at the terminal the entire line in which the pointer is located. It provides a ready means of determining the location of the pointer after a search completes and before you give any editing commands. (The V command combines the two commands 0LL.) You can also type the V command after an editing command to allow proofreading of the results. No arguments are allowed with the V command. The location of the pointer does not change.

5.6.6 Text Modification Commands

You can use the following commands to insert, relocate, and delete text in the text buffer.

5.6.6.1 Insert – The Insert (I) command is the basic command for inserting text. EDIT inserts the text you supply at the location of the pointer, then places the pointer after the last character of the new text.

The syntax of the command is:

Itext$

No arguments are allowed with the insert command, and the text string is limited only by the size of the text buffer and the space available. All characters except ESCAPE are legal in the text string. ESCAPE terminates the text string.

NOTE

If you forget to type the I command, the text will be executed as commands.

EDIT automatically protects against overflowing the text buffer during an insert. If the I command is the first command in a multiple command line, EDIT ensures that there will be enough space for the insert to be executed at least once. If repetition of the command exceeds the available memory, an error message prints.

The following example illustrates the use of the I command.

```
*IMOV           #BUFF,R2
MOV             #LINE,R1
MOVB            -1(R2),R0$$
*
```

This command inserts the text at the current location of the pointer and leaves the pointer positioned after R0.

DIGITAL recommends that you insert large amounts of text into the file in small sections rather than all at once. This way, you are less vulnerable to loss of time and effort due to machine failure or human error. This is the recommended technique for inserting large amounts of text:

1. Open the file with the EB command
2. Insert or edit a few pages of text
3. Insert a unique text string (like ????) to mark your place
4. Use the Exit command to preserve the work you have done so far
5. Start again, using the F command to search for the unique string you used to mark your place
6. Delete your marker and continue editing.

By using this procedure, you reduce your loss (should there be a machine or human error) to the few pages of text on which you just worked.

5.6.6.2 Delete – The Delete (nD) command is a character-oriented command that deletes n characters in the text buffer beginning at the current location of the pointer. The syntax of the command is:

nD

If you do not specify n, EDIT deletes the character immediately following the pointer. Upon completion of the D command, EDIT positions the pointer immediately before the first character following the deleted text. Table 5-9 lists each argument for the D command and its effect.

Table 5-9 Delete Command Arguments

Argument	Meaning
n	Deletes n characters following the pointer. Places the pointer before the first character following the deleted text.
-n	Deletes n characters preceding the pointer. Places the pointer before the first character following the deleted text.
0	Deletes the current line up to the pointer. The position of the pointer does not change (equivalent to 0K).
/	Deletes the text between the pointer and the end of the buffer. Positions the pointer at the end of the buffer (equivalent to /K).
=	Deletes -n characters, where n equals the length of the last text argument used.

The following examples illustrate the use of the D command.

```
*-2D$$
```

This command deletes the two characters immediately preceding the pointer.

```
*B$FMOV R1$=D$$
```

This command string deletes the text string MOV R1. (=D in combination with a search command deletes the indicated text string.)

Assume the text buffer contains the following:

```
ADD    R1,(R2)+
CLR    ↑@R2
```

The following command deletes the current line up to the pointer:

```
*0D$$
```

The buffer now contains:

```
ADD    R1,(R2)+
↑@R2
```

5.6.6.3 Kill – The Kill (nK) command removes n lines of text (including the carriage return and line feed characters) from the page buffer, beginning at the pointer and ending with the nth end-of-line. The syntax of the command is:

```
nK
```

EDIT places the pointer at the beginning of the line following the deleted text. Table 5-10 describes each argument and its effect upon the Kill command.

Table 5-10 Kill Command Arguments

Argument	Meaning
n	Removes the character string (including the carriage return/line feed combination) beginning at the pointer and ending at the nth end-of-line.
-n	Removes the character string beginning at the nth end-of-line preceding the pointer and ending at the pointer. Thus, if the pointer is at the center of a line, the modifier -1 deletes one and one-half lines preceding it.
0	Removes the current line up to the pointer (equivalent to 0D).
/	Removes the characters beginning at the pointer and ending with the last line in the text buffer (equivalent to /D).

The following examples use the K command.

```
*2K$$
```

This command deletes lines starting at the current location of the pointer and ending at the second carriage return/ line feed combination.

Assume the text buffer contains the following:

```
ADD    R1,(R2)+
CLR    @R2
MOVB   6(R1),@R2
```

This command removes the characters beginning at the pointer and ending with the last line in the text buffer:

```
*/K$$
```

The buffer now contains:

```
ADD    R1,(R2)+
CLR
```

Kill and Delete commands perform the same function, except that Kill is line-oriented and Delete is character-oriented.

5.6.6.4 Change – The Change (nC) command changes a specific number of characters following the pointer. The syntax of the command is:

```
nCtext
```

A C command is equivalent to a Delete command followed by an Insert command. You must insert a text object following the nC command. Table 5-11 lists each argument and its effect upon the C command.

Table 5-11 Change Command Arguments

Argument	Meaning
n	Replaces n characters following the pointer with the specified text. Places the pointer after the inserted text.
-n	Replaces n characters preceding the pointer with the specified text. Places the pointer after the inserted text.
0	Replaces the current line up to the pointer with the specified text. Places the pointer after the inserted text (equivalent to 0X).
/	Replaces the text beginning at the pointer and ending with the last character in the buffer. Places the pointer after the inserted text (equivalent to /X).
=	Replaces -n characters with the indicated text string, where n represents the length of the last text argument used.

The size of the text is limited only by the size of the text buffer and the space available. All characters are legal except ESCAPE, which terminates the text string.

If the C command is to be executed more than once (i.e., it is enclosed in angle brackets) and if there is enough space available for the command to be entered, it will be executed at least once (provided it appears first in the command string). If repetition of the command exceeds the available memory, an error message prints.

The following examples use the C command.

```
*5C#VECT$$
```

This command replaces the five characters to the right of the pointer with #VECT.

Assume the text buffer contains the following:

```
CLR     @R2
MOV     5(R1),@R2
```

The next command replaces the current line up to the pointer with the specified text.

```
*0CADDB$$
```

The buffer now contains:

```
CLR     @R2
ADDB    5(R1),@R2
```

You can use =C with a Get command to replace a specific text string. Here is an example:

```
*GFIFTY:$=CFIVE:$
```

This command finds the occurrence of the text string FIFTY: and replaces it with the text string FIVE:.

5.6.6.5 eXchange – The eXchange (nX) command is similar to the change command except that it changes lines of text, instead of a specific number of characters. The syntax of the command is:

nXtext

The nX command is identical to an nK command followed by an Insert command. Table 5-12 lists each argument and its effect upon the eXchange command.

Table 5-12 eXchange Command Arguments

Argument	Meaning
n	Replaces n lines including the carriage return and line feed characters following the pointer. Places the pointer after the inserted text.
-n	Replaces n lines including the carriage return and line feed characters preceding the pointer. Positions the pointer after the inserted text.
0	Replaces the current line up to the pointer with the specified text. Positions the pointer after the inserted text (equivalent to 0C).
/	Replaces the text beginning at the pointer and ending with the last character in the buffer with the specified text (equivalent to /C). Positions the pointer after the inserted text.

All characters are legal in the text string except ESCAPE, which terminates the text.

If the X command is enclosed within angle brackets to allow more than one execution, and if there is enough memory space available for the X command to be entered, EDIT executes it at least once (provided it is first in the command string). If repetition of the command exceeds the available memory, an error message prints.

The following example uses the X command.

```
*2XADD   R1,(R2)+
 CLR     @R2
 $$
*
```

This command exchanges the two lines to the right of the pointer with the text string.

5.6.7 Utility Commands

During the editing session, you can store text in external buffers and subsequently restore this text when you need it later on in the editing session. The following sections describe the commands that perform this function.

5.6.7.1 Save – The Save (nS) command lets you store text in an external buffer called a save buffer (described previously in Section 5.5), and subsequently insert it in several places in the text.

The syntax of the command is:

nS

The Save command copies n lines, beginning at the pointer, into the save buffer. The S command operates only in the forward direction; therefore, you cannot use a negative argument. The Save command destroys any previous contents of the save buffer; however, EDIT does not change the location of the pointer or the contents of the text buffer.

If you specify more characters than the save buffer can hold, EDIT prints ?EDIT-F-INSUFFICIENT MEMORY. None of the specified text is saved.

For example, assume the text buffer contains the following assembly-language subroutine:

```
; SUBROUTINE MSGTYP
; WHEN CALLED, EXPECTS R0 TO POINT TO AN
; ASCII MESSAGE THAT ENDS IN A ZERO BYTE,
; TYPES THAT MESSAGE ON THE USER TERMINAL

MSGTYP:   TSTB    (R0)          ; DONE?
          BEQ     MDONE         ; YES-RETURN
MLOOP:    TSTB    @#177564      ; NO-IS TERMINAL READY?
          BPL     MLOOP         ; NO-WAIT
          MOVB    (R0)+,@#177566 ; YES PRINT CHARACTER
          BR      MSGTYP        ; LOOP
MDONE:    RTS     PC            ; RETURN
```

The following command stores the entire subroutine in the save buffer (assuming the pointer is at the beginning of the buffer):

```
*12S$$
```

You can insert the contents of the save buffer into a program whenever you choose by using the Unsave command.

5.6.7.2 Unsave – The Unsave (U) command inserts the entire contents of the save buffer into the text buffer at the pointer and leaves the pointer positioned following the inserted text. You can use the U command to move blocks of text or to insert the same block of text in several places. Table 5-13 lists the U commands and their meanings.

Table 5-13 U Command and Arguments

Command	Meaning
U	Inserts the contents of the save buffer into the text buffer.
0U	Clears the save buffer and reclaims the area for text.

The only argument the U command accepts is 0.

The contents of the save buffer are not destroyed by the Unsave command (only by the 0U command) and can be unsaved as many times as desired. If the Unsave command causes an overflow of the text buffer, the ?EDIT-F-INSUFFICIENT MEMORY error message prints, and the command does not execute.

For example:

```
*U$$
```

This command inserts the contents of the save buffer into the text buffer.

Table 5-14 M Command and Arguments

Command	Meaning
M/command string/	Stores the command string in the macro buffer.
0M or M//	Clears the macro buffer and reclaims the area for text.

The slash (/) represents the delimiter character. The delimiter is always the first character following the M command, and can be any character that does not appear in the macro command string itself.

Starting with the character following the delimiter, EDIT places the macro command string characters into its internal macro buffer until the delimiter is encountered again. At this point, EDIT returns to command mode. The macro command does not execute the macro string; it merely stores the command string so that the Execute Macro (EM) command can execute later. The Macro command does not affect the contents of the text or save buffers.

All characters except the delimiter are legal macro command string characters, including single ESCAPEs to terminate text commands. All commands, except the M and EM commands, are legal in a command string macro.

In addition to using the 0M command, you can type the M command immediately followed by two identical characters (assumed to be delimiters) and two ESCAPE characters to clear the macro buffer.

The following examples illustrate the use of the M command.

```
*M//$$
```

This command clears the macro buffer.

```
*M/GR0$-C1$/$$
```

This command stores a macro to change R0 to R1.

NOTE

Be careful to choose infrequently-used characters as macro delimiters; use of frequently-used characters can lead to inadvertent errors. For example:

```
*M GMOV R0$=CADD R1$ $$
?EDIT-F-No file open for input
```

In this case, it was intended that the macro be GMOV R0$=CADD R1$ but since the delimiter character (the character following the M) is a space, the space following MOV is used as the second delimiter, terminating the macro. EDIT then returns an error when it interprets the R as a Read command.

5.6.7.4 Execute Macro – The Execute Macro (nEM) command executes the command string previously stored in the macro buffer by the M command.

The syntax of the command is:

nEM

The argument (n) must be positive. The macro is executed n times and returns control to the next command in the original command string.

The following example uses the EM command.

```
*M/BGR0$-C1$/$$
*B1000EM$$
?EDIT-F-Search failed
*
```

This command sequence executes the macro stored in the previous example. EDIT prints an error message when it reaches the end of the buffer. (This macro changes all occurrences of R0 in the text buffer to R1.)

```
*IMOV PC,R1$2EMICLR @R2$$
*
```

This command inserts MOV PC,R1 into the text buffer, then executes the command in the macro buffer twice before inserting CLR @R2 into the text buffer.

5.6.7.5 Edit Version – The Edit Version (EV) command displays the version number of the editor in use on the console terminal.

The command is:

```
EV
```

This example displays the running version of EDIT:

```
*EV$$
V03.36
*
```

5.6.7.6 Upper- and Lower-Case Commands – If you have an upper- and lower-case terminal as part of your hardware configuration, you can take advantage of the upper- and lower-case capability of this terminal. Two editing commands, EL and EU, permit this.

When the editor is first started with the EDIT command, upper-case mode is assumed; all characters you type are automatically translated to upper case. To allow processing of both upper- and lower-case characters, enter the Edit Lower command. For example:

```
*EL$$
*i You can enter text and commands in UPPER and lower case.$$
*
```

The editor now accepts and echoes upper- and lower-case characters received from the keyboard and prints text on the terminal in upper and lower case.

To return to upper-case mode, use the Edit Upper command:

```
*EU$$
```

Control also reverts to upper-case mode upon exit from the editor (with EX or CTRL/C).

Note that when you issue an EL command, you can enter EDIT commands in either upper or lower case. Thus, the following two commands are equivalent:

```
*GTEXT$=Cnew text$V$$

*gTEXT$=cnew text$v$$
```

The editor automatically translates (internally) all commands to upper case independent of EL or EU.

NOTE

When you use EDIT in EL mode, make sure that text arguments you specify in search commands have the proper case. The command GTeXt$, for example, will not match TEXT, text, or any combination other than TeXt.

5.7 THE DISPLAY EDITOR

In addition to all functions and commands mentioned thus far, the editor can use VT-11 and VS-60 display hardware that may be part of the system configuration (GT40, GT44, DECLAB 11/40, DECLAB 11/34). The most obvious feature is the ability to use the display screen rather than the console terminal for printing all terminal input and output. Another feature is that the top of the display screen functions like a window into the text buffer. When all the features of the display editor are in use, a 12 in. screen displays text as shown in Figure 5-1.

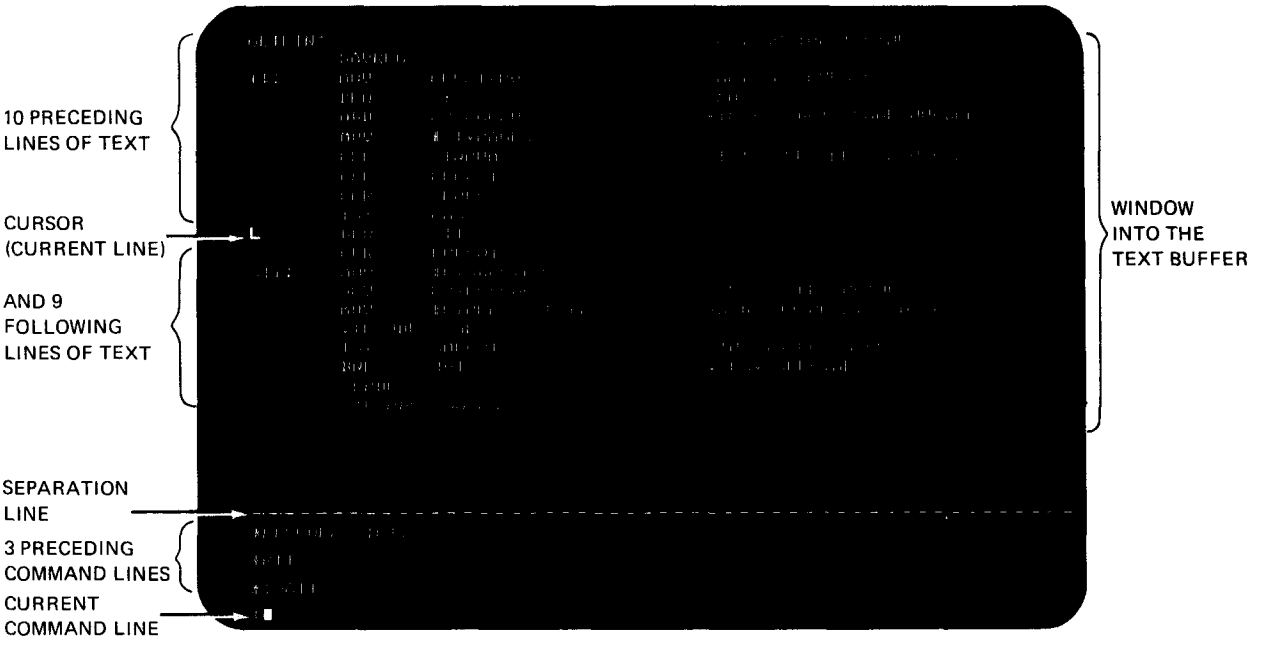


Figure 5-1 Display Editor Format, 12 in. Screen

The major advantage is that you can now see immediately where the pointer is. The pointer appears between characters on the screen as a blinking L-shaped cursor and you can see it easily. Remember that pressing the (RET) key causes both a carriage return and a line feed character to be inserted into the text. Note that if the pointer is placed between a carriage return and line feed, it appears in an inverted position at the beginning of the next line.

In addition to displaying the current line (the line containing the cursor), the 15 lines of text preceding the current line and the 14 lines following it are also in view on a 17 in. screen. Each time you execute a command string (with a double ESCAPE), EDIT refreshes this portion of the screen so that it reflects the results of the commands you just performed.

The lower section of the 17 in. screen contains eight lines of editing commands. The command line you are currently entering is last, preceded by the most recent command lines. A horizontal line of dashes separates this section from the text portion of the screen. As you enter new command lines, previous command lines scroll upward off the command section so that only eight command lines are ever in view.

A 12 in. screen displays 20 lines of text and 4 command lines.

5.7.1 Using the Display Editor

The display features of the editor are automatically invoked whenever the system scroller is in use (a monitor GT ON command is in effect) and you start the editor. However, if the system does not contain display hardware, the display features are not enabled.

Providing that the system does contain display hardware and that you wish to employ the screen during the editing session, you can activate it in one of two ways, whether or not the display is in use. All editing commands and functions previously discussed in this chapter are valid for use.

1. If the scroller is in use (the GT ON monitor command is in effect), EDIT recognizes this and automatically uses the screen for display of text and commands. However, it rearranges the scroller so that a window into the text buffer appears in the top two-thirds of the screen, while the bottom third displays command lines. This arrangement is shown in Figure 5-1.

   You can use the Edit Console command to return the scroller to its normal mode so that text and commands use the full screen, and the window is eliminated.

   The command is:

   ```
   EC
   ```

   This example uses the EC command:

   ```
   *BAEC2L$$
   ```

   This command lists the second and third lines of the current buffer on the screen; there is no window into the text buffer at this point.

   EDIT ignores subsequent EC commands if the window into the text buffer is not being displayed.

   To recall the window, use the Edit Display command:

   ```
   ED
   ```

   The screen is again arranged as shown in Figure 5-1.

2. Assume the scroller is not is use (the GT ON command is not in effect). When you call EDIT with the .EDIT command, an asterisk appears on the console terminal. Use the ED command at this time to provide the window into the text buffer; however, commands continue to be echoed to the console terminal.

   When you use ED in this case, it must be the first command you issue; otherwise, it becomes an illegal command (since the memory used by the display buffer and code, amounting to over 600 words, is reclaimed as working space). You cannot use the display again until you load a fresh copy of EDIT.

   While the display of the text window is active, EDIT ignores ED commands.

   Typing the EC command clears the screen and returns all output to the console terminal.

NOTE

After an editing session that uses the ED command is over, clear the screen by typing the EC command or by returning to the monitor and using the monitor RESET command. Failure to do this may cause unpredictable results.

5.7.2 Setting the Editor to Immediate Mode

An additional mode is available in EDIT to provide easier and faster interaction during the editing session. This mode is called immediate mode and combines the most-used functions of the text and command modes – namely, repositioning the pointer and deleting and inserting characters.

You can only use immediate mode when the VT-11 display hardware is active and the editor is running. Enter it by typing two ESCAPEs (only) in response to the command mode asterisk:

```
*$$
```

The editor responds by echoing an exclamation point on the screen.

```
!
```

The exclamation character remains on the screen as long as control is in immediate mode.

Once you enter immediate mode, you can use only the commands in Table 5-15. Any other commands or characters are treated as text to be inserted. None of these commands echoes, but the text appearing on the screen is constantly refreshed and updated during the editing process.

To return control to the display editor's normal command mode at any time while in immediate mode, type a single ESCAPE. The editor responds with an asterisk and you can proceed using all normal editing commands. (Immediate mode commands you type at this time will be accepted as command mode input characters.) To return control to the monitor while in immediate mode, type ESCAPE to return to command mode, then type CTRL/C followed by two ESCAPEs.

Table 5-15 Immediate Mode Commands

Command	Meaning
CTRL/N	Advances the pointer (cursor) to the beginning of the next line (equivalent to A).
CTRL/G	Moves the pointer (cursor) to the beginning of the previous line (equivalent to -A).

(Continued on next page)

Table 5-15 (Cont.) Immediate Mode Commands

Command	Meaning
CTRL/D	Moves the pointer (cursor) forward by one character (equivalent to J).
CTRL/V	Moves the pointer (cursor) back by one character (equivalent to -J).
RUBOUT or DELETE	Deletes the character immediately preceding the pointer (cursor) (equivalent to -D).
ESCAPE or ALTMODE	Single character returns control to command mode; double character directs control to immediate mode.
Any character other than those above	Inserts the character as text positioned immediately before the pointer (cursor) – equivalent to I.

5.8 EDIT EXAMPLE

The following example illustrates the use of some of the EDIT commands to change a program stored on the device DK:. Sections of the terminal output are coded by letter and corresponding explanations follow the example.

```
A  EDIT/OUTPUT:TEST2.MAC TEST1.MAC
*R$$
*/L$$
;TEST PROGRAM

START:  MOV     #1000,SP        ;INITIALIZE STACK
        MOV     #MSG,R0         ;POINT R0 TO MESSAGE
B       JSR     PC,MSGTYP       ;PRINT IT
        HALT                    ;STOP
MSG:    .ASCII/IT WORKS/
        .BYTE 15
        .BYTE 12
        .BYTE 0

C
*B$1J$5D$$
*GPROGRAM$$
D  *0L$$
;PROGRAM*I TO TEST SUBROUTINE MSGTYP. TYPES
E  ;"THE TEST PROGRAM WORKS"
;ON THE TEMI\IM\RMINAL$$
F   G.ASCII/$$
*$CTHE TEST PROGRAM WORKS$$
*P.BYTE^X
G  *G.BYTE 0$V$$
        .BYTE 0
```

```
*I
          .END
$B/L$$
;PROGRAM TO TEST SUBROUTINE MSGTYP. TYPES
;"THE TEST PROGRAM WORKS"
;ON THE TERMINAL

START:  MOV     #1000,SP        ;INITIALIZE STACK
        MOV     #MSG,R0         ;POINT R0 TO MESSAGE
        JSR     PC,MSGTYP       ;PRINT IT
        HALT                    ;STOP
MSG:    .ASCII/THE TEST PROGRAM WORKS/
        .BYTE 15
        .BYTE 12
        .BYTE 0
        .END


*EX$$

.
```

A Calls the EDIT program and prints *. The input file is TEST1.MAC; the output file is TEST2.MAC. Reads the first page of input into the buffer.

B Lists the buffer contents.

C Places the pointer at the beginning of the buffer. Advances the pointer one character (past the ;) and deletes the TEST.

D Positions the pointer after PROGRAM and verifies the position by listing up to the pointer.

E Inserts text. Uses RUBOUT to correct typing error.

F Searches for .ASCII/ and changes IT WORKS to THE TEST PROGRAM WORKS.

G Types CTRL/X to cancel the P command. Searches for .BYTE 0 and verifies the location of the pointer with the V command.

H Inserts text. Returns the pointer to the beginning of the buffer and lists the entire contents of the buffer.

I Closes the input and output files after copying the current text buffer as well as the rest of the input file into the output file. EDIT returns control to the monitor.

5.9 EDIT ERROR CONDITIONS

The editor prints an error message whenever a detectable error condition occurs. EDIT checks for three general types of error conditions: 1) syntax errors, 2) execution errors and, 3) macro execution errors. This section describes the error message form for each type of error condition.

Before it executes any commands, EDIT first scans the entire command string for errors in command syntax, such as illegal arguments or an illegal combination of commands. If the editor finds an error of this type, it prints a message of this form:

?EDIT-F-Message; no command(s) executed

You should retype the command.

If a command string is syntactically correct, EDIT begins execution. Execution errors, such as buffer overflow or input and output errors, can still occur. In this case, EDIT prints a message of the form:

```
?EDIT-F-Message
```

EDIT executes all commands preceding the one in error. It does not execute the command in error or any commands that follow it.

When an error occurs during execution of a macro, EDIT prints a message of the form:

```
?EDIT-F-Message in macro; no command(s) executed
or
?EDIT-F-Message in macro
```

Most errors are syntax errors. These are usually easy to correct before execution.

The *RT-11 System Message Manual* contains a complete list of the EDIT error messages, along with recommended corrective action for each error.

PART IV
UTILITY PROGRAMS

The following chapters describe in detail the system programs available to you as an RT-11 user. You can take advantage of nearly all of the capabilities of the RT-11 system by using the keyboard monitor commands, which are described in Chapter 4. However, it is the system utility programs (and not the monitor itself) that actually perform many of the system's functions. When you issue a monitor COPY command, for example, it is a system utility program (PIP, DUP, or FILEX, in this case) that performs the copy operation. Part IV of this manual, Utility Programs, explains how to carry out utility operations, those not performed directly by the monitor, by running a specific system utility program instead of using the keyboard monitor commands. It is not necessary to have an understanding of the material contained in Part IV in order to use the RT-11 system. However, the information in Part IV may be of interest to you if you have experience with a previous version of RT-11, or if you are a systems programmer and need to perform certain functions with the utility programs that are not available with the keyboard monitor commands. Note that the syntax the Command String Interpreter requires for input and output specifications is different from the syntax you use to issue a keyboard monitor command. Chapter 6, the Command String Interpreter, describes the general syntax of the specification string that the system utility programs accept, and explains certain conventions and restrictions. Read this chapter carefully before you use any of the system utility programs directly, and bear in mind that there are many differences between issuing a monitor command and running a utility program. Chapters 7 through 15 describe the system utility programs themselves.

CHAPTER 6
COMMAND STRING INTERPRETER

The Command String Interpreter (CSI) is the part of the RT-11 system that accepts a line of ASCII input, usually from you at the console terminal, and interprets it as a string of input specifications, output specifications, and options for use by a system utility program. To call a utility program, respond to the dot (.) printed by the keyboard monitor by typing R followed by a program name and a carriage return. This example shows how to call the directory program (DIR):

```
.R DIR
```

The Command String Interpreter prints an asterisk (*) at the left margin on the terminal, indicating that it is ready to accept a list of specifications and options. The following section describes the syntax of the specifications and options you can enter.

6.1 COMMAND STRING INTERPRETER SYNTAX

Once you have started a system program, you must enter the appropriate information before any operation can be performed. You type a specification string in response to the prompting asterisk. The specifications are in the following general syntax:

```
output-filespecs/option=input-filespecs/option
```

(A few system programs – EDIT and PATCH, for example – require you to enter this information slightly differently. Complete instructions are provided in the appropriate chapters.)

In all cases, the syntax for output-filespec is:

```
dev:filnam.typ[n], . . . dev:filnam.typ[n]
```

The syntax for input-filespec is:

```
dev:filnam.typ , . . . dev:filnam.typ
```

The syntax for /option is:

```
/o:oval or /o:dval.
```

where

dev: represents either a logical device name or a physical device name, which is a 2- or 3-character name from Table 3-1.

If you do not supply a device name, the system uses device DK:. DK:, or whatever device you specify for the first file in a list of input or output files, applies to all the files in that input or output list, until you supply a different device name. For example:

```
*DT1:FIRST.OBJ,LP:=TASK.1,RK1:TASK.2,TASK.3
```

This command is interpreted as follows:

```
*DT1:FIRST.OBJ,LP:=DK:TASK.1,RK1:TASK.2,RK1:TASK.3
```

File FIRST.OBJ is stored on device DT1:. File TASK.1 is stored on default device DK:. Files TASK.2 and TASK.3 are stored on device RK1:. Notice that file TASK.1 is on device DK:. It is the first file in the input file list and the system uses the default device DK:. Device DT1: applies only to the file on the output side of the command.

filnam.typ — represents the name of a file (consisting of one to six alphanumeric characters followed optionally by a dot and a zero to three character file type). No spaces or tabs are allowed in the file name or file type. As many as three output and six input files are allowed.

[n] — is an optional declaration of the number of blocks (n) you need for an output file; n is a decimal number (<65,535) enclosed in square brackets immediately following the output filnam.typ to which it applies.

/o:oval or /o:dval. — represents one or more options whose functions vary according to the program you are using (refer to the option table in the appropriate chapter); oval is either an octal number or one to three alphanumeric characters (the first of which must be alphabetic) that the program converts to Radix-50 characters; dval. is a decimal number followed by a decimal point.

This manual uses the /o:oval construction throughout, except for the keyboard monitor commands, where all values are interpreted as decimal (unless indicated otherwise) and the decimal point after a value is not necessary. However, the /o:dval. format is always valid. Generally, these options and their associated values, if any, should follow the device and file name to which they apply.

If the same option is to be repeated several times with different values (e.g., /L:MEB/L:TTM/L:CND) you can abbreviate the line as /L:MEB:TTM:CND. You can mix octal, Radix-50, and decimal values.

= — If required, is a delimiter that separates the output and input fields. You can use the < sign in place of the = sign. You can omit the separator entirely if there are no output files.

6.2 PROMPTING CHARACTERS

Table 6-1 summarizes the characters RT-11 prints either to indicate that the system is awaiting your response or to specify which job (foreground or background) is producing output.

Table 6-1 Prompting Characters

Character	Explanation
.	The keyboard monitor is waiting for a command.
^	When the console terminal is being used as an input file, the uparrow (or circumflex) prompts you to enter information from the keyboard. Typing a CTRL/Z marks the end-of-file.
>	The > character identifies (only if a foreground job is active) which job, foreground or background, is producing the output that currently appears on the console terminal. Each time output from the background job is to appear, B> prints first, followed by the output. If the foreground job is to print output, F> prints first.
*	The current system utility program is waiting for a line of specifications and options.

CHAPTER 7
PERIPHERAL INTERCHANGE PROGRAM (PIP)

The peripheral interchange program (PIP) is a file transfer and file maintenance utility program for RT-11. You can use PIP to transfer files between any of the RT-11 devices (listed in Table 3-1) and to merge, rename, and delete files.

7.1 CALLING AND USING PIP

To call PIP from the system device, respond to the dot (.) printed by the keyboard monitor by typing:

```
R PIP (RET)
```

The Command String Interpreter prints an asterisk at the left margin of the terminal and waits for you to type a command string. If you type only a carriage return at this point, PIP prints its current version number and prompts you again for a command string. You can type CTRL/C to halt PIP and return control to the monitor when PIP is waiting for input from the console terminal. You must type two CTRL/Cs to abort PIP at any other time. To restart PIP, type R PIP or REENTER followed by a carriage return in response to the monitor's dot. Chapter 6, Command String Interpreter, describes the general syntax of the command line that PIP accepts. You can type as many as six input file names, but only one output file name is allowed. You can put command options at the end of the command string or type them after any file name in the string. Operations involving magtape are an exception to this rule because the /M option is device dependent, and has a different meaning when you specify it on the input or output side of a command line. Type any number of options in a command line, as long as only one operation (insertion, deletion, etc.) is represented. You can, however, combine copy and delete operations on one line. If you specify a command involving random access devices for which the output specification is the same as the input specification, PIP does not move any files. However, it can change the creation dates on the files if you use /T, or it can rename the files if you use /R.

Since PIP performs file transfers for all RT-11 data formats (ASCII, object, and image), it does not assume file types for either input or output files. You must explicitly specify all file types, where file types are applicable.

On random access devices, such as disks and DECtape, PIP operations retain a file's creation date. If the file's creation date is 0, PIP gives it the current system date. However, in transfers to and from magtape and cassette, PIP always gives files the current system date.

You can use all variations of the wildcard construction for the input file specifications in the PIP command line (Section 4.3 describes wildcard usage). Output file specifications cannot contain embedded wildcards. If you use any wild character in an input file specification, the corresponding output file name or file type must be an asterisk. The concatenate copy operation is an exception to this rule because it does not allow wildcards in the output specification. These two lines are examples of wildcard usage:

```
**.B=A%B.MAC

*B.*=A%B.MAC
```

The first command string is legal. The second generates an error message because the file name field of the input specification contains a wildcard and the output specification is not *.

The following command, for example, deletes all files with the file type .BAK (regardless of their file names) from device DK:.

```
**.BAK/D
```

The next command renames all files with a .BAK file type (regardless of file names) so that these files now have a .TST file type (maintaining the same file names).

```
**.TST=*.BAK/R
```

PIP performs operations on files in the order in which you specify them in the command string. However, if the specification contains a wildcard, PIP operates on the files in the order in which they appear in the device directory. PIP ignores system files with the file type .SYS unless you also use the /Y option. PIP prints the error message ?PIP-W-NO.SYS ACTION if you omit the /Y option on a command that would operate on .SYS files.

PIP ignores all files with the file type .BAD unless you explicitly specify both the file name and file type in the command string. PIP does not print a warning message when it does not include .BAD files in an operation. Because of the way PIP handles .BAD files, you cannot use a wildcard (*.BAD) to perform any operation on them.

This example transfers all files, including system files, (regardless of file name or file type) from device DK: to device RK1:. It does not transfer .BAD files.

```
*RK1:*.*/Y=*.*
```

7.2 PIP OPTIONS

Certain options permit you to perform various operations with PIP. Table 7-1 summarizes the operations that PIP performs. If you do not specify an option, PIP assumes that the operation is a file transfer in image mode. The following sections are organized by function. Operations involving magtape and cassette are discussed first because these operations are treated uniquely by PIP. The other functions (copy, delete, rename, log, and query) are described next. Explanations of the options are arranged alphabetically in the discussions of the appropriate functions.

Table 7-1 PIP Options

Option	Section	Explanation
/A	7.2.2.2	Copies files in ASCII mode, ignoring nulls and rubouts. It converts to 7-bit ASCII and treats CTRL/Z (32 octal) as the logical end-of-file on input (the default copy mode is image).
/B	7.2.2.3	Copies files in formatted binary mode (the default copy mode is image).
/C	7.2.2.4	Can be used with another option. It causes PIP to include only files with the current date in the specified operation.
/D	7.2.3	Deletes input files from a specific device. Note that PIP does not automatically query before it performs the operation. If you combine /D with a copy operation, PIP performs the delete operation after the copy completes. This option is illegal in an input specification with magtape.
/G	7.2.2.5	Ignores any input errors that occur during a file transfer and continues copying.
/K:n	7.2.2.6	Makes n copies of the output files to LP:, TT:, or PC:.
/M:n	7.2.1	You can use /M:n when I/O transfers involve either cassette or magtape. (See Section 7.2.1, Operations Involving Magtape or Cassette.)

(Continued on next page)

Table 7-1 (Cont.) PIP Options

Option	Section	Explanation
/N	7.2.2.7	Does not copy or rename a file if a file with the same name exists on the output device. This option protects you from accidentally deleting a file. This option is illegal for magtape and cassette in the output specification.
/O	7.2.2.8	Deletes a file on the output device if you copy a file with the same name to that device. The delete operation occurs before the copy operation. This option is illegal for magtape and cassette in the output specification.
/P	7.2.2.9	Copies or deletes all files except those you specify.
/Q	7.2.6	Use only with another operation. The /Q option causes PIP to print the name of each file to be included in the operation you specify. You must respond with a Y to include a particular file.
/R	7.2.4	Renames the file you specify. This operation is illegal for magtape and cassette.
/S	7.2.2.10	Copies files one block at a time.
/T	7.2.2.11	Puts the current date on all files you copy or rename, unless the current date is 0. This option is illegal for magtape and cassette; operations involving those devices always use the current date.
/U	7.2.2.12	Copies and concatenates all files you specify.
/W	7.2.5	Prints on the terminal a log of copy, rename, and delete operations.
/Y	7.2.2.13	Includes .SYS files in the operation you specify. You cannot modify or delete these files unless you use the /Y option.

7.2.1 Operations Involving Magtape and Cassette

PIP handles operations that involve magtape and cassette devices differently from operations that involve random access devices, such as disks and DECtape. That is because magtape and cassette are sequential access devices. This means that files are stored serially, one after another, on the device and that there is no directory at the beginning of each device that lists the files and gives their location. Because of the serial nature of tape, you can access only one file at a time on each device unit. Avoid commands that specify the same device unit number for both the input and output files – they are illegal. The /M:n option is designed to make operations that involve magtape and cassette more efficient. This option lets you specify different tape handling procedures for PIP to follow. The following sections outline the operations that involve magtape and cassette and describe the different procedures for using these devices that you can specify with the /M:n option. Remember that when you use the /M:n option, n is interpreted as an octal number. You must use n. (n followed by a decimal point) to represent a decimal number.

7.2.1.1 Using Cassette – The cassette is an inexpensive auxiliary storage medium. Cassettes are typically used to store data such as text files or source programs. Clear plastic leader indicates the beginning-of-tape (BOT) and physical end-of-tape (EOT). A special control file marks the end of current data and indicates where

operation. You can also use it to specify a special procedure for tape handling during cassette operations with PIP. The following operations are valid for use with cassettes: /A, /B, /C, /D, /G, /M, /P, /Q, /R, /S, /U, /W, and /Y.

These options are illegal with cassettes: /K, /N, /O, /R, and /T. If you omit the /M:n option in a cassette operation, the cassette rewinds before each operation. Using /M:0 has the same effect. The character n represents a count of the number of files from the present position on the cassette. Note that the /M:n option has a different meaning for cassette and magtape. Section 7.2.1.2 describes how to use /M:n with magtape.

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

1. If n is 0:

   The cassette rewinds and PIP searches for the file you specify. If you specify more than one file, or if you use a wildcard in the file specification, the cassette rewinds before PIP searches for each file.

2. If n is a positive integer:

   PIP starts from the cassette's present position and searches for the file you specify. If PIP does not find the file by the time it reaches the nth file from its starting position, it uses the nth file for the read operation. Note that if PIP's starting position is not the beginning of the cassette, it is possible that PIP will not find the file you specify, even though it does exist on the tape.

3. If n is a negative integer:

   The cassette rewinds, then PIP follows the procedure outlined in step 2 above.

For cassette write (copy to tape) operations, the /M:n option initiates these procedures:

1. If n is 0:

   The cassette rewinds and PIP writes the file you specify starting at the logical end-of-tape (LEOT) position. PIP automatically deletes any file it finds along the way that has the same name and file type as the file you specify.

2. If n is a positive integer:

   PIP starts from the cassette's present position and searches n files ahead, deleting along the way any file it finds that has the same name and file type as the file you specify. If it does not reach LEOT before it reaches the nth file from its starting position, it enters the file you specify over the nth file and deletes any files beyond it on the tape. If PIP reaches LEOT before it reaches the nth file, it writes the file you specify at the end-of-tape.

3. If n is a negative integer:

   The cassette rewinds, then PIP follows the same procedure outlined in step 2 above.

If you are copying a file to cassette and reach the physical end-of-tape before the copy completes, PIP automatically continues the file on another cassette. The cassette device handler prints the CTn: PUSH REWIND OR MOUNT NEW VOLUME message. If you want to halt the copy operation at this point, push the cassette rewind button. The tape rewinds, PIP prints an error message, and then PIP prompts you for a new command. However, if you want to continue the file on another cassette, remove the first cassette and put another initialized cassette in its place. The new cassette rewinds immediately. PIP then continues copying the file. The continued part of the file has the same file name and file type as the first part of the file, but PIP adds 1 to its sequence number to show that it is a continued file. Make sure you have a supply of initialized cassettes handy for cassette copy operations; you cannot interrupt the copy operation to initialize a cassette when PIP is waiting for a new volume. The following example shows a copy operation that fills one cassette and continues to another.

```
*CT1:*.*=RK:RK*.SYS,%%.SYS/Y/W/M:1
 Files copied:
RK:RKMNSJ.SYS  to CT1:RKMNSJ.SYS
CT1: PUSH REWIND OR MOUNT NEW VOLUME
RK:RKMNFB.SYS  to CT1:RKMNFB.SYS
RK:DT.SYS      to CT1:DT.SYS
RK:DP.SYS      to CT1:DP.SYS
RK:DX.SYS      to CT1:DX.SYS
RK:RF.SYS      to CT1:RF.SYS
RK:RK.SYS      to CT1:RK.SYS
RK:DM.SYS      to CT1:DM.SYS
RK:DS.SYS      to CT1:DS.SYS
RK:TT.SYS      to CT1:TT.SYS
RK:LP.SYS      to CT1:LP.SYS
RK:CR.SYS      to CT1:CR.SYS
RK:MT.SYS      to CT1:MT.SYS
RK:MM.SYS      to CT1:MM.SYS
RK:NL.SYS      to CT1:NL.SYS
RK:PC.SYS      to CT1:PC.SYS
RK:EL.SYS      to CT1:EL.SYS
RK:CT.SYS      to CT1:CT.SYS
RK:BA.SYS      to CT1:BA.SYS
*
```

A directory listing of the second cassette shows that the first file, RKMNFB.SYS, is continued from a previous tape. (The number of blocks in a cassette directory listing is not meaningful; it really represents the total of sequence numbers in the directory.)

```
.DIRECTORY CT1:
 15-Apr-77
RKMNFB.SYS     1 15-Apr-77      DT     .SYS    0 15-Apr-77
DP     .SYS    0 15-Apr-77      DX     .SYS    0 15-Apr-77
RF     .SYS    0 15-Apr-77      RK     .SYS    0 15-Apr-77
DM     .SYS    0 15-Apr-77      DS     .SYS    0 15-Apr-77
TT     .SYS    0 15-Apr-77      LP     .SYS    0 15-Apr-77
CR     .SYS    0 15-Apr-77      MT     .SYS    0 15-Apr-77
MM     .SYS    0 15-Apr-77      NL     .SYS    0 15-Apr-77
PC     .SYS    0 15-Apr-77      EL     .SYS    0 15-Apr-77
CT     .SYS    0 15-Apr-77      BA     .SYS    0 15-Apr-77
 18 Files, 1 Blocks
```

If you are reading a file from cassette that is continued on another volume, the cassette handler also prints the CTn: PUSH REWIND OR MOUNT NEW VOLUME message when it reaches the end of the first tape. To abort the operation, push the cassette rewind button; PIP then issues an error message and prompts for a new command. To continue the read operation, remove the first cassette and mount the second one in its place. The second cassette rewinds immediately and PIP searches for a file with the correct name and sequence number. PIP repeats the new volume message if it does not find the correct file. The following example copies a file that is continued on a second cassette.

```
*RK1:*.*=CT1:RKMNFB.SYS/Y/W
 Files copied:
CT1: PUSH REWIND OR MOUNT NEW VOLUME
CT1:RKMNFB.SYS to RK1:RKMNFB.SYS
*
```

If you type a double CTRL/C during any output operation to cassette, PIP does not write a sentinel file at the end of the tape. Consequently, you cannot transfer any more data to the cassette unless you follow one of these two recovery procedures:

1. First, rewind the cassette. Then, transfer all good files from the interrupted cassette to another cassette and initialize the interrupted cassette as the following example shows. Use any arbitrarily large number for /M:n.

```
*CT1:*.*=CT0:DMPX.MAC,EXAMP.FOR/M:1000
*^C
.R DUP
*CT0:/Z/Y
*
```

2. Determine the sequential number of the file that was interrupted and use the /M:n construction to enter a replacement file (either a new file or a dummy) over the interrupted file. PIP writes the replacement file and a sentinel file (LEOT) after it. The following example assumes the bad file is the fourth file on the cassette.

```
*CT0:DUMMY.FIL=DT0:GLOBAL.MAC/M:-4
*^C

.DIRECTORY CT0:
 19-Apr-77
DMPX   .MAC      0 19-Apr-77      MATCH .BAS      0 19-Apr-77
EXAMP  .FOR      0 19-Apr-77      DUMMY .FIL      0 19-Apr-77
 4 Files, 0 Blocks
```

   A directory listing of the cassette shows three files and the replacement file.

To copy multiple files to a cassette with a wildcard command, use the following:

```
*CTn:*.*=dev:*.*/M:1
```

Continue to mount new cassettes in response to the PUSH REWIND OR MOUNT NEW VOLUME message. Do not abort the process at any time (using two CTRL/Cs) since continuation files may not be completed and no sentinel file will be written on the cassette.

To read multiple files from a cassette, use a command like the following one. Use any arbitrarily large number for /M:n.

```
*dev:*.*=CTn:*.*/M:1000
```

Whenever PIP detects a continued volume, the PUSH REWIND OR MOUNT NEW VOLUME message appears, until the entire file has been copied (assuming that you mount each sequential cassette in response to each occurrence of the message). When PIP copies the final section of a continued file, it returns to command level. To copy the remaining files on that cassette, reissue the command:

```
*dev:*.*=CTn:*.*/M:1000
```

Repeat the process as often as necessary to copy all files. Do not abort the process at any time (using two CTRL/Cs) since continuation files may not be completed.

7.2.1.2 Using Magtape – Magnetic tape is a convenient auxiliary storage medium for large amounts of data. Magtapes are often used as backup for disks. Reflective strips indicate the beginning and end of the tape. A special label (an EOF1 or EOV1 tape label) followed by two tape marks indicates the end of current data and indicates where new data can begin. The following PIP options are valid for use with magtape: /A, /B, /C, /G, /M, /P, /Q, /S, /U, /W, and /Y. These options are illegal with magtape: /D, /K, /N, /O, /R, and /T. The /M:n option lets you direct the tape operation; you can move the tape and perform an operation at the point you specify. The /M:n option can be different for the output and input side of the command line. Since the option applies to the device and not to the files, you can specify one /M:n option for the output file and one for the input files. The /M:n option has a different meaning for cassette and magtape. Section 7.2.1.1 describes how to use /M:n with cassette.

Sometimes PIP begins an operation at the current position. To determine the current position, the magtape handler backspaces from its present position on the tape until it finds either an EOF indicator or the beginning of tape, whichever comes first. PIP then begins the operation with the file immediately following the EOF or BOT. The magtape handler also has a special procedure for locating a file with sequence number n:

1. If the file sequence number is greater than the current position, PIP searches the tape in the forward direction.
2. If the file sequence number is more than one file before the current position, or if the file sequence number is less than five files from the beginning-of-tape (BOT), the tape rewinds before PIP begins its search.
3. If the file sequence number is at the current position, or if it is one file past the current position, PIP searches the tape in the reverse direction.

Whenever you fetch or load a new copy of the magtape handler, the tape position information is lost. The "new" handler searches backwards until it locates either BOT or a label from which it can learn the position of the tape. It then operates normally, according to steps 1, 2, and 3 described above.

If you omit the /M:n option, the tape rewinds between each operation. Using /M:0 has the same effect as omitting /M:n. When n is positive, it represents the file sequence number. When n is negative, it represents an instruction to the magtape handler.

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

1. If n is 0:

   The tape rewinds and PIP searches for the file you specify. If you specify more than one file, the tape rewinds before each search. If the file specification contains a wildcard, the tape rewinds only once and then PIP copies all the appropriate files.

2. If n is a positive integer:

   PIP goes to file sequence number n. If the file it finds there is the one you specify, PIP copies it, Otherwise, PIP prints the ?PIP-F-FILE NOT FOUND message. If you use a wildcard in the file specification PIP goes to file sequence number n and then begins to search for matching files.

3. If n is -1:

   PIP starts the search at the current position. Note that if the current position is not the beginning of the tape, it is possible that PIP will not find the file you specify, even though it does exist on the tape.

For magtape write (copy to tape) operations, the /M:n option initiates these procedures:

1. If n is 0:

   The tape rewinds before PIP copies each file. PIP prints a warning message if it finds a file with the same name and file type as the input file and does not perform the copy operation.

2. If n is a positive integer:

   PIP goes to the file sequence number n and enters the file you specify. If PIP reaches LEOT before it finds file sequence number n, it prints the ?PIP-F-FILE SEQUENCE NUMBER NOT FOUND message. If you specify more than one file or if you use a wildcard in the file specification, the tape does not rewind before PIP writes each file, and PIP does not check for duplicate file names.

3. If n is -1:

   PIP goes to the LEOT and enters the file you specify. It does not check for duplicate file names.

4. If n is -2:

   The tape rewinds between each copy operation. PIP enters the file at LEOT or at the first occurrence of a duplicate file name.

If PIP reaches the physical end-of-tape before it completes a copy operation, it cannot continue the file on another tape volume. Instead, it deletes the partial file by backspacing and writing a logical end-of-tape over the file's header label. You must restart the operation and use another magtape.

If you type two CTRL/Cs during any output operation to magtape, PIP does not write a logical end-of-tape at the end of the data. Consequently, you cannot transfer any more data to the tape unless you follow one of the two following recovery procedures. In addition, if the magtape handler was loaded (with the monitor LOAD command), you must unload it before you can access the tape.

1. Transfer all good files from the interrupted tape to another tape and initialize the interrupted tape in the following manner:

```
*dev1:*.*=dev0:*.*
^C
.R DUP
*dev0:/Z/Y
```

2. Determine the sequential number of the file that was interrupted and use the /M:n construction to enter a replacement file (either a new file or a dummy) over the interrupted file. PIP writes the replacement file and a good LEOT after it. The following example assumes the bad file is the fourth file on the tape:

```
*dev0:file.new=file.dum/M:4
```

7.2.2 Copy Operations

The following sections describe the types of copy operation that PIP can perform. PIP copies files in image, ASCII, and binary format. Other options let you change the date on the files, access .SYS files, combine files, and perform other similar operations. PIP automatically allocates the correct amount of space for new files in copy operations (except for concatenation). For block-replaceable devices, PIP stores the new file in the first empty space large enough to accommodate it. If an error occurs during a copy operation, PIP prints a warning message, stops the copy operation, and prompts you for another command. You cannot copy .BAD files unless you specifically type each file name and file type.

7.2.2.1 Image Mode – If you enter a command line without an option, PIP copies files onto the destination device in image mode. Note that you cannot reliably transfer memory image files to or from paper tape, or to the line printer or console terminal. PIP can image-copy ASCII and binary data but it does not do any of the data checking described in Sections 7.2.2.2 or 7.2.2.3.

The following command makes a copy of the file named XYZ.SAV on device DK: and assigns it the name ABC.SAV. (Both files exist on device DK: following the operation.)

```
*ABC.SAV=XYZ.SAV
```

The next example copies from DK: all .MAC files whose names are three characters long and begin with A. PIP stores the resulting files on DX1:.

```
*DX1:*.*=A%%.MAC
```

7.2.2.2 ASCII Mode (/A) – Use the /A option to copy files in 7-bit ASCII mode. PIP ignores nulls and rubouts in an ASCII mode file transfer. PIP treats CTRL/Z (32 octal) as logical end-of-file if it encounters that character in the input file. The following command copies F2.FOR from device DK: onto device DT1: in ASCII mode and assigns it the name F1.FOR.

```
*DT1:F1.FOR=F2.FOR/A
```

7.2.2.3 Binary Mode (/B) – Use the /B option to transfer formatted binary files (such as .OBJ files produced by the assembler or the FORTRAN compiler and .LDA files produced by the linker). The following command, for example, transfers a formatted binary file from the paper-tape reader to device DK: and assigns it the name FILE.OBJ.

```
*DK:FILE.OBJ=PC:/B
```

When performing formatted binary transfers, PIP verifies checksums and prints a warning if a checksum error occurs. If there is a checksum error and you did not use /G to ignore the error, PIP does not perform the copy operation. You cannot copy library files with the /B option; PIP prints the ?PIP-F-LIBRARY FILE NOT COPIED message. Copy them in image mode.

7.2.2.4 The Newfiles Option (/C) – Use the /C option in the command line if you want to copy only those files with the current date. Specify /C only once in the command line. It applies to all the file specifications in the entire command. The following command copies (in ASCII mode) all files named ITEM1.MAC that also have the current date. It also copies the file ITEM2.MAC, if it has the current date, from DK: to DT2:. It combines all these files under the name NN3.MAC.

```
*DT2:NN3.MAC=ITEM1.MAC*/C,ITEM2.MAC/A/U
```

The next command copies all files with the current date (except .SYS and .BAD files) from DK: to DX1:. This is an example of an efficient way to back up all new files after a session at the computer.

```
*DX1:*.*=*.*/C
```

7.2.2.5 The Ignore Errors Option (/G) – Use the /G option to copy files, but ignore all input errors. This option forces a single-block transfer, which you can invoke at any other time with the /S option. Use the /G option if an input error occurred when you tried to perform a normal copy operation. The procedure can sometimes recover a file that is otherwise unreadable. If an error still occurs, PIP prints the ?PIP-W-INPUT ERROR message and continues the copy operation. The following command, for example, copies the file TOP.SAV in image mode from device DT1: to device DK: and assigns it the name ABC.SAV.

```
*ABC.SAV=DT1:TOP.SAV/G
```

The next command copies files F1.MAC and F2.MAC in ASCII mode from device DT1: to device DT2:. This command creates one file with the name COMB.MAC, and ignores any errors that occur during the operation.

```
*DT2:COMB.MAC=DT1:F1.MAC,F2.MAC/A/G/U
```

7.2.2.6 The Copies Option (/K:n) – The /K:n option directs PIP to generate n copies of the file you specify. The only legal output devices are the console terminal, the line printer, and paper-tape punch. Normally, each copy of the file begins at the top of a page; copies are separated by form feeds.

```
*LP:=FOO.LST/K:3
```

This command, for example, prints three copies of the listing file, FOO.LST, on the line printer.

7.2.2.7 Noreplace Option (/N) – The /N option prevents execution of a copy or rename operation if a file with the same name as the output file already exists on the output device. This option is illegal for magtape and cassette. The following example uses the /N option.

```
*DX0:CT.SYS=DK:CT.SYS/Y/N
?PIP-W-Output file found, no operation performed DK:CT.SYS
*
```

The file named CT.SYS already exists on DX0:, and the copy operation does not proceed.

7.2.2.8 The Predelete Option (/O) – The /O option deletes a file on the output device if you copy a file with the same name to that device. PIP deletes the file on the output device before the copy operation occurs. Normally, PIP deletes a file of the same name after the copy completes. This option is illegal for magtape and cassette. The following example uses the /O option.

```
*RK1:TEST1.MAC=DT2:TEST.MAC/O
```

If a file named TEST1.MAC already exists on RK1:, PIP deletes it before copying TEST.MAC from DT2: to TEST1.MAC on RK1:.

7.2.2.9 The Exclude Option (/P) – The /P option directs PIP to transfer all files except the ones you specify.

```
*DT0:*.*=DX1:*.MAC/P
```

This command, for example, directs PIP to transfer all files from DX1: to DT0: except the .MAC files.

7.2.2.10 The Single-block Transfer Option (/S) – The /S option directs PIP to copy files one block at a time. On some devices, this operation increases the chances of an error-free transfer. You can combine the /S option with other PIP copy options. For example:

```
*RK1:TEST.MAC=RK0:TEST.MAC/S
```

PIP performs this transfer one block at a time.

7.2.2.11 The Setdate Option (/T) – This option causes PIP to put the current date on all files it transfers, unless the current date is 0. Normally, PIP preserves the existing file creation date on copy and rename operations. This option is invalid for operations involving magtape and cassette because PIP always uses the current date for

```
**.*=*.*/Y/T
```

Note that the command shown above changes only the dates; PIP does not move or change the files in any other way.

7.2.2.12 The Concatenate Option (/U) – To combine more than one file into a single file, use the /U option. This option is particularly useful to combine several object modules into a single file for use by the linker or librarian. PIP does not accept wildcards on the output specification. The following examples use the /U option.

```
*DK:AA.OBJ=DT1:BB.OBJ,CC.OBJ,DD.OBJ/U
```

The command shown above transfers files BB.OBJ, CC.OBJ and DD.OBJ to device DK: as one file and assigns this file the name AA.OBJ.

```
*DT3:MERGE.MAC=DT2:FILE2.MAC,FILE3.MAC/A/U
```

This command merges ASCII files FILE2.MAC and FILE3.MAC on DT2: into one ASCII file named MERGE.MAC on device DT3:.

7.2.2.13 The System Files Option (/Y) – Use the /Y option if you need to perform an operation on system files (.SYS). For example:

```
**.*=DT3:*.*/G/Y
```

This command copies to device DK:, in image mode, all files (including .SYS files) from device DT3:. Because of the /G option, PIP ignores any input errors.

7.2.3 The Delete Operation (/D)

Use the /D option to delete one or more files from the device you specify. Note that PIP does not automatically query you before it performs the operation; you must use /Q. Remember to use the /Y option to delete .SYS files. You cannot delete .BAD files unless you name each one specifically, including file name and file type. You can specify only six files in a delete operation unless you use wildcards. You must always indicate a file specification in the command line. A delete command consisting only of a device name (dev:/D) is invalid. The delete option is also illegal for magtape. The following examples illustrate the delete operation.

```
*FILE1.SAV/D
```

The command shown above deletes FILE1.SAV from device DK:.

```
*DX1:*.*/D
?PIP-W-No .SYS action
*
```

The command shown above deletes all files from device DX1: except those with a .SYS or .BAD file type. If there is a file with a .SYS file type, PIP prints a warning message to remind you that this file has not been deleted.

```
**.MAC/D
```

This command deletes all files with a .MAC file type from device DK:.

7.2.4 The Rename Operation (/R)

Use the /R option to rename a file you specify as input, giving it the name you specify in the output specification. You must supply an equal number of input and output files that reside on the same device. PIP prints an error message if the command specifications are not valid. Use the /Y option with /R if you rename .SYS files. You cannot use /R with magtape or cassette.

The rename command is particularly useful when a file on disk or DECtape contains bad blocks. By renaming the file, giving it a .BAD file type, you can ensure that the file permanently resides in that area of the device. Thus, the system makes no other attempts to use the bad area. Once you give a file a .BAD file type, you cannot move it during a compress operation. You cannot rename .BAD files unless you specifically indicate both the file name and file type. The following examples illustrate the rename operation.

```
*DT1:F1.MAC=DT1:F0.MAC/R
```

The command shown above renames F0.MAC to F1.MAC on device DT1:.

```
*DX1:OUT.SYS=DX1:CT.SYS/Y/R
```

This command renames file CT.SYS to OUT.SYS.

7.2.5 The Logging Operation (/W)

When you use the /W option, PIP prints a list of all files copied, renamed, or deleted. The /W option is useful if you do not want to take the time to use the query mode (the /Q option, described in Section 7.2.6), but you do want a list of the files operated on by PIP.

PIP prints the log for an operation on the terminal beneath the command line. This example shows logging with the delete operation.

```
*DX1:*.*/D/W
?PIP-W-No .SYS action
 Files deleted:
DX1:TEST.MAC
DX1:FIX463.SAV
DX1:GRAPH.BAK
DX1:DMPX.MAC
DX1:MATCH.BAS
DX1:EXAMP.FOR
DX1:GRAPH.FOR
DX1:GLOBAL.MAC
DX1:PROSEC.MAC
DX1:KB.MAC
DX1:EXAMP.MAC
*
```

7.2.6 The Query Option (/Q)

Use the /Q option with another PIP operation to list all files and to confirm individually which of these files should be processed. Typing a Y (or any string that begins with Y) followed by a carrige return causes the named file to be processed; typing anything else excludes the file. The following example deletes files from DX1:.

```
*DX1:*.*/D/Q
 Files deleted:
DX1:FIX463.SAV?
DX1:GRAPH.BAK ? Y
DX1:DMPX.MAC  ?
DX1:MATCH.BAS ?
DX1:EXAMP.FOR ?
DX1:GRAPH.FOR ? Y
DX1:GLOBAL.MAC? Y
DX1:PROSEC.MAC? Y
DX1:KB.MAC    ?
DX1:EXAMP.MAC ?
*
```

CHAPTER 8
DEVICE UTILITY PROGRAM (DUP)

The device utility program (DUP) is a device maintenance utility program you can use with the RT-11 system. DUP creates files on file-structured RT-11 devices (disks, DECtape, magtape, and cassette). It can also extend files on certain file-structured devices (disks and DECtape), and it can compress, image copy, initialize, or boot RT-11 file-structured devices. DUP does not operate on non-file-structured devices (line printer, card reader, terminal, and paper tape).

8.1 CALLING AND USING DUP

To call DUP from the system device, respond to the dot (.) printed by the keyboard monitor by typing:

R DUP (RET)

The Command String Interpreter prints an asterisk (*) at the left margin of the terminal and waits for a command string. If you enter only a carriage return in response to the asterisk, DUP prints its current version number. You can type CTRL/C to halt DUP and return control to the monitor when DUP is waiting for input from the console terminal. You must type two CTRL/Cs to abort DUP at any other time. The /S, /T, and /C operations, however, lock out the CTRL/C command until the operation completes; these three operations cannot be interrupted with CTRL/C. To restart DUP, type R DUP or REENTER in response to the monitor's dot. Chapter 6, Command String Interpreter, describes the general syntax of the command line that DUP accepts. DUP accepts only one input file specification and one output file specification in the command line.

8.2 DUP OPTIONS

Certain options are available for use with DUP. These options are divided into two categories: 1) Action and 2) Mode. Action options cause specific operations to occur. You can use these options alone or with valid mode options. Usually, you can specify only one action option at a time. Mode options modify action options. Table 8-1 illustrates whch mode options you can use with a particular action option.

Table 8-1 DUP Options and Categories

Action	Mode
C	W,Y
I	W,Y
K	W,F,H
O	W,Y
S	W,X,Y
T	W,Y
U	W
V	W
Z	W,B,N,R,V,Y

Note that /V can be either an action or a mode option, depending on how you use it.

You can use DUP action options to create files, copy devices, scan for bad blocks, perform a bootstrap operation, and so on. You can use the DUP mode options to modify the action options, where necessary. The following sections describe the various DUP options and give examples of typical uses. Table 8-2 summarizes the options you can use with DUP.

Table 8-2 DUP Options

Option	Section	Explanation
/B	8.2.11.4	Use with /Z to write files with the file type .BAD over any bad blocks DUP finds on the disk to be initialized.
/C:m[:n]	8.2.1	Creates a file on the device you specify; m represents the starting block number (in octal) and n represents the size of the file in blocks.
/F	8.2.3	Use with the /K option to output the file name containing the bad block together with the relative block number of the bad block in the file.
/H	8.2.3	Use with the /K option to read the bad block, write to the bad block, and then read it again. This operation does not destroy information already stored on the device.
/I[:rstart :rstop :wstart]	8.2.2	Copies the image of a disk to another disk or magtape or from magtape to disk. The arguments :rstart, :rstop, and :wstart represent block numbers.
/K[:start [:stop]]	8.2.3	Scans a device for bad blocks and outputs the octal address of the logical bad blocks to the output device. The arguments :start and :stop represent block numbers.
/N:n	8.2.11.1	Use with /Z to set the number of directory segments you require if you do not want the default size; n is an integer in the range 1-37 (octal).
/O	8.2.4	Boots the device or file you specify.
/R[:RET]	8.2.11.3	Use with /Z to scan the RK06 device for bad blocks and to create a replacement table on the disk for any bad blocks DUP finds. If you use :RET, DUP retains the replacement table that is already on the disk and does not pre-scan the disk for bad blocks.
/S	8.2.5	Compresses a disk (or DECtape) onto itself or onto another disk (or DECtape); the output device, if any, must be initialized.
/T:n	8.2.6	Extends an existing file by the number of blocks you indicate by :n.
/U	8.2.7	Writes the bootstrap portion of the monitor file in blocks 0 and 2-5 of the target device.
/V[:VOL]	8.2.8, 8.2.11.2	Prints the user ID and owner name. Use it with /Z (as a mode option) to insert a user ID and owner name in block 1 of the initialized disk, or in the VOL1 header block on magtape (not applicable for cassette). Using /V:VOL as an action option causes only the ID and owner name to be changed, and does not initialize the device (not applicable for cassette).

(Continued on next page)

Table 8-2 (Cont.) DUP Options

Option	Section	Explanation
/W	8.2.9	Use with any action option (but only one) to initiate an operation and then pause. This is useful on small, single-disk systems because it lets you replace the system device with another disk before performing an operation.
/X	8.2.5	Use with /S to inhibit automatic booting of the system device when it is compressed.
/Y	8.2.10	Use with /C, /I, /O, /S, /T, or /Z to inhibit the dev:/xxxx ARE YOU SURE? message and the FOREGROUND JOB LOADED, CONTINUE? message and ensure immediate execution of the operation.
/Z[:n]	8.2.11	Initializes the directory of the device you specify. The size of the directory defaults to the standard RT-11 size; use :n to allocate extra directory words for each entry beyond the default.

8.2.1 The Create Option (/C:m[:n])

The /C option creates a file with a specific name, location, and size on the block-replaceable device that you specify. This command is useful to recover files that have been deleted. The /C option only creates a directory entry for the file. It does not store any data in the file. You must specify both the file name and file type of the file to be created. The syntax of the command is:

```
filespec=/C:m[:n]
```

where

filespec	represents the device, file name and file type of the file to be created.
:m	represents the numeric value, in octal, of the starting block of the file to be created.
:n	represents the size of the file in blocks. If you do not supply a value for n, DUP creates a 1-block file.

You can use the /C option to cover bad blocks on a disk by creating a file with a file type .BAD to cover the bad area.

Use /C also to recover accidentally-deleted files. In this case, use DIR to obtain a listing of the device. Use the /E and the /Q options in DIR; obtain a separate listing with each one. DIR lists files, tentative files, empty areas, and the sizes of all areas. You can then assign a file name to the area that contains the data you lost.

You can also use DUP to set aside a file on disk without performing any input or output operations on the file.

When you use the /C option, make sure that the area in which the file is to be created is empty. If there are more blocks in the empty than the file you are creating needs, DUP attempts to put the extra blocks in empties that are contiguous to the file you are creating. If there is not enough room in contiguous empties, the error message ?DUP-F-ILLEGAL CONTIGUOUS FILE prints and DUP does not create the file. The /C option checks for duplicate file names. If the file name you specify already exists on the device, DUP issues an error message and does not create a second file with the same name.

This is an example of a command that uses /C:

```
*DK1:FILE.MAC=/C:140:3
```

This command creates a file named FILE.MAC consisting of blocks 140, 141, and 142 on device DK1:.

8.2.2 The Image Copy Option (/I)

The /I option copies block for block from one device to another. (This operation is not applicable for magtape or cassette.) If DUP encounters a bad block, it prints an error message. However, it retries the operation and performs the copy one block at a time. If only one error message prints, you can assume that the transfer completed correctly. The /I option is often used to copy one disk to another without changing the file structure or location of files on the device. In this case, it is an added convenience that you do not have to copy a boot block to the device. You can also copy disks that are not in RT-11 format, if they have no bad blocks.

Qualifiers to the /I option let you specify the blocks to be read from the input device; you can also specify a starting block number on the output device for the write operation. The syntax of the command is:

output-device:=input-device:/I[:rstart:rstop:wstart]

where

:rstart	represents the starting block number on the input device for the read operation.
:rstop	represents the ending block number on the input device for the read operation.
:wstart	represents the starting block number on the output device for the write operation.

The command string must include an input and an output specification; there is no default device. If you need to specify a block number, you must supply all three block values. The /I operation does not copy to or from a device that has logical bad blocks. (Physical bad blocks can be logically replaced or covered, as Sections 8.2.11.3 and 8.2.11.4 describe.) If one device is smaller than the other, DUP copies only the number of blocks of the smaller device.

You can copy blocks between disk and magtape with /I. DUP stores the data on the tape, formatting it in 1K word blocks. It is possible to store only one disk image on a magtape, regardless of the size of the tape.

The following examples use the /I option. The file name A is not significant; it is a dummy file name required by the Command String Interpreter.

```
*RK1:A=RK0:/I

RK1:/Copy are you sure?
```

The command shown above copies all blocks from DK: to RK1:.

```
*RK1:A=RK0:/I:0:500:501

RK1:/Copy are you sure?Y
```

This command copies blocks 0-500 from RK0: to RK1: starting at block 501

Sometimes devices (disks and DECtapes) are manufactured with bad blocks, or they develop bad blocks as a result of use and age. You can use the /K option to scan a device and locate bad blocks on it. DUP prints

the absolute block number of those blocks on the device that return hardware errors when DUP tries to read them. If you specify an output device (only TT: and LP: are valid), DUP prints the bad block report on that device. Remember that block numbers are octal and the first block on a device is block 0. If DUP finds no bad blocks, it prints only the header. A complete scan of a disk pack takes from one to several minutes depending on the size of the device. It does not destroy data that is stored on the device.

DUP reads only one block at a time when it scans a disk for bad blocks. Errors can occur on a multi-block copy even if DUP does not detect any with /K. Copy the data to a scratch disk with the /I option to discover any other bad blocks. You should scan a device for bad blocks before using /S to compress the device; if a read error occurs during a compress operation, the device may become unuseable.

You can scan selected portions of a device by specifying a beginning and ending block number. The syntax of this command is:

```
[output-device:=]input-device:/K[:start[:stop]]
```

where

:start represents the block number of the first block to be scanned.

:stop represents the block number of the last block to be scanned.

If you specify only a starting block number, DUP scans from the block you specify to the end of the device. You cannot specify an ending block number unless you also specify a starting block number.

If the device to be scanned has files on it, you can use /F with the /K option to print the name of the file containing the bad block together with the relative block number within the file that is bad.

You can use /H with /K to read the bad block, write to the bad block, and then read it again. If the block is still bad, DUP reports a HARD error. If the block recovers, DUP reports a SOFT error. This procedure does not destroy data already stored on the device.

The following command line uses the /K option to scan the entire disk, RK1:.

```
*RK1:/K/F
BAD BLOCKS FILENAME      REL BLK TYPE
  6615     EMPTY1.TST     6547   HARD
  6645     EMPTY2.TST     6577   HARD
  7255     EMPTY3.TST     7207   HARD
```

8.2.4 The Boot Option (/O)

The /O option can perform two operations: 1) a hardware bootstrap of a specific device and 2) a bootstrap of a particular monitor file that does not affect the bootstrap blocks on the device. The command syntax for a device bootstrap is as follows:

```
dev:/O
```

This operation has the same results as a hardware bootstrap. Legal devices for the boot operation are DT0:, RK0:-RK7:, RF:, SY:, DK:, DP0:-DP7:, DX0:-DX1:, DM0:-DM7:, and DS0:-DS7:.

Use the following syntax to boot the monitor you specify without changing the bootstrap on the device.

```
dev:monitor-name/O
```

This makes it easy for you to switch from one monitor to another. Whether bootstrapping a specific monitor or a specific device, DUP checks to see if the bootstrap blocks are correctly formatted. If the boot operation you request is invalid for any reason, DUP prints an error message and waits for another command.

When you reboot with the /O option, you do not have to reenter the date and time of day with the monitor DATE and TIME commands. However, the clock does lose a few seconds during the reboot.

The following command reboots the RT-11 system under the single-job monitor:

```
*RK0:RKMNSJ.SYS/O

RT-11SJ     V03.01
```

Notice in this command that the device you specify must be the same device type as the first two characters of the monitor file indicate. Because of this restriction on the monitor-name bootstrap operation, the following command is illegal:

```
*RK0:DXMNFB.SYS/O
```

However, the next command is a valid one:

```
*RK0:RKMNFB.SYS/O
```

8.2.5 The Squeeze Option (/S)

Use the /S option to compress a device (disk or DECtape) onto itself or onto another disk or DECtape. To do this, DUP moves all the files to the beginning of the device, producing a single, unused area after the group of files. The squeeze operation does not change the bootstrap blocks of a device. The output device you specify, if any, must be an initialized device. If you specify an output device, DUP does not query you for confirmation before it performs the operation. If you do not specify an output device, DUP prints the ARE YOU SURE? message and waits for your response before proceeding. You must type Y followed by a carriage return to execute the command. Since it is critical to perform an error-free squeeze operation, be sure to scan a device (with /K) before you use /S.

The /S option does not move files with .BAD file types. This feature prevents you from reusing bad blocks that occur on a disk. You can rename files containing bad blocks, giving them a .BAD file type, and DUP then leaves them in place when you execute a /S. DUP inserts files before and after .BAD files until the space between the last file it moved and the .BAD file is smaller than the next file to be moved. If an error occurs during a squeeze operation, DUP continues the operation, performing it one block at a time. If only one error message prints, you can assume that the operation completed correctly.

The syntax of the command is:

```
[output-device=]input-device/S
```

Do not use /S on the system device (SY:) when a foreground job is loaded. A ?DUP-F-CANNOT WRITE SY: WHILE FJOB LOADED error message results if you attempt this and DUP ignores the /S operation. You must unload the foreground job before using the /S option.

NOTE

If you perform a compress operation on the system device

to prevent system crashes that can occur when the monitor file is moved.

You can use /X with /S to suppress the automatic reboot and leave DUP running. However, you should use /X only if you are certain that the monitor file will not move. Even then, you should reboot the system when the squeeze operation completes if the device handlers have moved. If you specify the /X option but for some reason the USR cannot be made resident, DUP reboots the system anyway. If you use /X and the system is not rebooted, the ?DUP-W-REBOOT message prints. This is a warning message; it is for your information only.

The following examples use the /S command:

```
*SY:/S

 SY:/Squeeze are you sure?Y

RT-11SJ      V03.01
```

The command shown above compresses the files on the system device and reboots the system when the compress operation completes.

```
*DT1:A=DT2:/S
```

This command transfers all the files from device DT2: to device DT1:, leaving DT2: unchanged. The file name A is not significant; it is a dummy file name required by the Command String Interpreter.

8.2.6 The Extend Option (/T:n)

Use the /T option to extend the size of a file. The syntax of the command is:

filespec/T:n

where

filespec — represents the device, file name, and file type of the file to be extended.

n — represents the number of blocks to add to the file.

You can extend a file in this manner only if it is followed by an unused area at least n blocks long. Any blocks not required by the extend operation remain in the unused area.

The following example uses the /T option:

```
*DT1:ZYZ.TST/T:100
```

This command assigns 100 more blocks to the file named ZYZ.TST on device DT1:.

8.2.7 The Bootstrap Copy Option (/U)

In order to use a disk as a system device, you must copy a bootstrap onto the disk. To do this, first make sure that the appropriate monitor file is stored on the disk. For a diskette system, for example, you could use the foreground/background monitor file called DXMNFB.SYS. If you copy the monitor file onto the diskette from another device, be careful not to rename it. DUP recognizes only standard RT-11 monitor file names in the bootstrap copy operation. Use the /U option to copy the bootstrap portion of the monitor file into absolute blocks 0 and 2-5 of the device. You can then use the /O option to boot the device.

1. Obtain a formatted disk. (Most disks and DECtapes are formatted by the manufacturer. However, the RT-11 System Generation Manual does outline the procedure for re-formatting an RK05 disk.)

2. Initialize the disk with /Z.

3. Copy files onto the disk.

4. Copy the monitor onto the disk.

5. Copy the monitor bootstrap onto the disk with /U.

The following example shows how to initialize a diskette, copy files to it, and write a bootstrap onto the diskette:

```
*DX1:/Z/Y
```

The command shown above (step 2 of the procedure described above) initializes the diskette.

```
*DX1:A=DX0:/S
```

This command, which combines steps 3 and 4, squeezes all the files from DX0: onto DX1:.

```
*DX1:A=DX1:DXMNFB.SYS/U
```

The last command (step 5) writes the bootstrap for the diskette foreground/background monitor onto the bootstrap blocks (blocks 0 and 2-5) of DX1:. The file name A is not significant; it is a dummy file name required by the Command String Interpreter.

8.2.8 The Volume ID Option (/V[:VOL])

You can use the /V option as an action option to print the volume ID of a device or to change the volume ID without initializing the device. The syntax of the command is:

```
device:/V[:VOL]
```

where

device: is the device whose volume ID you want to display or change.

If you specify only /V, the volume ID and owner name of the device you specify print out on the console terminal. If you specify /V:VOL, DUP assumes you need to change the volume ID and owner name. DUP prompts you for a volume ID:

```
VOL ID?
```

Respond with a volume ID that is up to 12 characters long for a block-replaceable device, or up to 6 characters long for magtape. Terminate your response with a carriage return. DUP then prompts for an owner name:

```
OWNER NAME?
```

Respond with an owner name that is up to 12 characters long for a block-replaceable device, or up to 10 characters long for magtape. Terminate your response with a carriage return. DUP ignores characters you type beyond the legal length. The /V:VOL command changes only the volume ID and owner name; it does not initialize the device. Section 8.2.11.2 describes how to use /V with the /Z option to initialize a device and write volume identification on it.

DUP stores the volume ID and owner name information in block 1 of a disk. The volume ID is stored in words 236-241 (decimal), the owner name is stored in words 242-247, and the format type, which is always RT11A, is stored in words 248-253. The remainder of block 1 (words 0-235 and 254-255) is reserved for the system to use. If you are initializing a magnetic tape, DUP stores the volume identification information in the VOL1 header block of the magtape. The volume ID is stored in bytes 5-10 and the owner name is stored in bytes 41-50. The first byte of the header block is byte 1; DUP stores VOL1 information up to byte 80.

The following example uses the /V:VOL option:

```
*RK1:/V:VOL/Y

 VOL ID? VOUCHERS

 OWNER NAME? PAYABLES
```

This command writes a new volume ID and owner name on device RK1:.

8.2.9 The Small, Single-disk System Option (/W)

The /W option is useful for small (8K), single-disk systems. It is a mode option that you can use with any of the action options. However, you can perform only one operation at a time. The /W option initiates execution of a command, but then pauses and prints the message CONTINUE?. At this time you can remove the system disk and mount the disk on which you actually want the operation to take place. When the new disk is loaded, type a Y followed by a carriage return to execute the operation. When the operation completes (except the /O operation, which boots the system), the "CONTINUE?" message again prints. Replace the system device and type a Y followed by a carriage return. The asterisk (*) prompt prints and DUP waits for you to enter another command. The following example uses the /W option:

```
*DX1:/K/F/W
CONTINUE?Y
BAD BLOCKS   TYPE   FILENAME      REL BLK
CONTINUE?Y
*
```

This command directs DUP to scan the disk for bad blocks. During the first pause, the system disk is removed and another disk is mounted. A Y is typed and the scan operation executes. During the second pause, the system disk is replaced and another Y is typed. DUP prompts for another command.

NOTE

It is not necessary to use the /W option to change disks if the USR can be made resident with the SET USR NOSWAP command. In this case you can change disks when the asterisk (*) prompt prints. Type and execute the command with the new disk in place. Replace the system disk when the next prompt prints.

There is one exception to the general usage of /W. You cannot use the /U option to write a bootstrap on another disk if you have a single-disk system with only 8K words of memory. Follow this procedure to write a bootstrap on another disk:

1. Make the USR resident:

```
.SET USR NOSWAP
```

2. Call the MDUP program (a program similar to DUP, but smaller):

```
.R MDUP
*
```

3. Change disks when MDUP prompts with an asterisk (*), as shown in step 2.

   The new disk must already have the monitor file stored on it. Then enter the /U command to copy the bootstrap, as this example shows:

```
*RK0:A=RK0:RKMNSJ.SYS/U
```

   When MDUP prints another asterisk, replace the system disk and type CTRL/C to return to the monitor.

8.2.10 The Noquery Option (/Y)

Use the /Y option to suppress the query messages that some commands print. The following options normally print the FOREGROUND JOB LOADED, CONTINUE? message if a foreground job is loaded when you issue one of these DUP commands: /C, /I, /O, /S, /T, and /Z. You must respond to the query message by typing Y followed by a carriage return for the operation to proceed. Some other options (/C, /I, /O, /S, /V, and /Z) print the ARE YOU SURE? message and wait for your response. If a foreground job is loaded and you specify one of these options, DUP combines the two query messages into one message and waits for your response. You can suppress all these messages and the pause associated with them by specifying /Y in the command string.

8.2.11 The Directory Initialization Option (/Z[:n])

You must initialize a device before you can store files on it. Use the /Z option to clear and initialize the directory of an RT-11 directory-structured device. The /Z operation must always be the first operation you perform on a new device after you receive it, formatted, from a manufacturer. After you use /Z, there are no files in the directory.

The syntax of the command is as follows:

```
device:/Z[:n]
```

In this command, the optional argument, n, is an octal number (greater than or equal to 1) indicating the change in size of each directory entry on a directory-structured device. The size of the directory determines the number of files that can be stored on a device. The system allows a maximum of 72 files per directory segment, and 31 directory segments per device. Each segment uses two blocks of available disk space. If you do not specify n, each entry is seven words long (for file name, creation date, and file length information). When extra words are allocated, the number of entries per directory segment decreases. The formula for determining the number of entries per directory segment is:

```
5-7/((# of extra words) +7)
```

For example, if you use /Z:1, you can make 63 entries per segment. RT-11 does not normally support non-standard directory formats. DIGITAL does not recommend altering the directory format. The number of directory segments in the directory defaults to the decimal value shown in Table 8-3 for the specified device.

Table 8-3 Default Directory Sizes

Device	Size (decimal) of Directory in Segments
RK05	16
DT	4
RF	4
DS	4
DP	31
DX	4
RK06	31

8.2.11.1 Changing Directory Segments (/N:n) – If you do not want the default size of the device, use /N with /Z to set the number of directory segments for entries in the directory. The syntax of the command is as follows:

/N:n

In this command, n represents the number of directory segments; n is an integer in the range 1-31.

The following example initializes the directory on device RK1: and allocates six directory segments.

```
*RK1:/Z/N:6

RK1:/Init are you sure?Y
```

8.2.11.2 Storing Volume ID (/V) – When you initialize a disk or magtape, DUP stores a default device ID of RT11A in block 1 of the device. You can use the /V option with /Z to insert a user ID and owner name in block 1 of the device. For example, the following command initializes device RK1: and prompts you for a volume ID and owner name. Section 8.2.8 illustrates these prompts and shows how to respond to them.

```
*RK1:/Z/V

RK1:/Init are you sure?Y

 VOL ID? VOUCHERS

 OWNER NAME? PAYABLES
```

8.2.11.3 Replacing Bad Blocks (/R[:RET]) – You can use the /R option with /Z if the device being initialized is an RK06. If DUP finds any bad blocks, it builds a replacement table of good blocks for them. The replacement table is stored in words 0-63 of block 1. (/R supports up to 32 bad blocks.) The RK06 then appears to have no bad blocks. Files that span the bad block use a replacement block instead of the bad block. The replacement blocks are located in the last cylinder of the disk. Speed of input and output operations decreases only when the replacement blocks for bad blocks are accessed. You can avoid this overhead by using the /B option (see Section 8.2.11.4) and not using bad block replacement. If DUP finds any bad blocks in a non-replaceable part of the disk, DUP reports that the disk is bad. When you initialize a device and want to retain the bad block replacement table that was created by a previous /R command, use /R:RET. The /R:RET option makes it easy to reinitialize an RK06 without rescanning it. After a disk is initialized with the /Z/R option combination, a scan of the disk with /K should reveal no bad blocks. If DUP finds a bad block during the /Z/R operation that is in blocks 0 through 5, it reports that the disk is not usable. If DUP finds a bad block that is not already marked on the disk as such,

it prints the ?DUP-W-UNMARKED BAD BLOCK message. This disk is not usable and must be reformatted by the manufacturer. If DUP finds bad blocks in the device directory, it prints a warning message. Bad blocks in the directory can cause considerable overhead and slow system performance on ENTER, LOOKUP, and CLOSE operations.

8.2.11.4 Covering Bad Blocks (/B) – To scan the disk for bad blocks and write files over them, use the /B option with /Z. For every bad block DUP encounters on the device, it creates a file called FILE.BAD to cover it. After the disk is initialized and the scan completed, the directory consists only of file FILE.BAD entries that cover the bad blocks. If DUP finds a bad block in the boot block or the directory, it prints an error message and the disk is not usable.

/R and /B are mutually exclusive options. You can use one or the other, but not both.

CHAPTER 9
THE DIRECTORY PROGRAM (DIR)

The directory program (DIR) performs a wide range of directory listing operations. It can list directory information about a specific device, such as the number of files stored on the device, their names, and their creation dates. DIR can list details about certain files, too, including their names, their file types, and their size in blocks. DIR can also print a device directory summary, and it can organize its listings in several ways, such as alphabetically or chronologically.

9.1 CALLING AND USING DIR

To call DIR from the system device, respond to the dot (.) printed by the keyboard monitor by typing:

R DIR (RET)

The Command String Interpreter prints an asterisk at the left margin of the terminal and waits for you to enter a command string. If you enter only a carriage return in response to the asterisk, DIR prints its current version number. You can type CTRL/C to halt DIR and return control to the monitor when DIR is waiting for input from the console terminal. You must type two CTRL/Cs to abort DIR at any other time. To restart DIR, type R DIR or REENTER in response to the monitor's dot. Chapter 6, Command String Interpreter, describes the general syntax of the command line that DIR accepts. Unless otherwise indicated, numeric arguments are interpreted as octal. Remember to put a decimal point after a decimal number to distinguish it from an octal number. Some of the DIR options accept a date as an argument in the command line. The syntax for specifying the date is:

dd.:mmm:yy.

where

dd.	represents the day (a decimal integer in the range 1-31).
mmm	represents the month (the first three characters of the name of the month).
yy.	represents the year (a decimal integer in the range 73-99).

You can specify only one input device and one output device, but you can specify up to six file names on the input device. The default device for output is the terminal. The default file type for an output file is .DIR. The default device for input is DK:. If you omit the input specification completely, DIR uses DK:*.*. If you do not supply an option, DIR performs the /L operation. Note that wildcards are valid with DIR for the input specification only.

Directory listings normally print on the terminal in two columns. Read the entries across the columns, moving from left to right, one row at a time. Directory listings that are sorted, however, are an exception to this. (Sorted directories are produced by /A, /R, and /S.) Read these listings by reading the left column from top to bottom, then reading the right column from top to bottom.

9.2 DIR OPTIONS

You can perform many different directory operations by specifying options in the DIR command line. Table 9-1 summarizes the operations these options permit you to perform with DIR. The following sections describe the various DIR options and give examples that use the options. The sections are arranged alphabetically by option.

Table 9-1 DIR Options

Option	Section	Explanation
/A	9.2.1	Lists the directory of the device you specify in alphabetical order by file name and type (this is the same as /S:NAM).
/B	9.2.2	Lists the directory of the device you specify, including file names and types, creation dates, starting block number in decimal, and the number of blocks in each file. For magtape, the starting block number is the file sequence number.
/C:n	9.2.3	Lists the directory in n columns; n is an integer in the range 1-9. The default value is two columns for normal listings and five columns for abbreviated listings.
/D[:date]	9.2.4	Includes in the directory listing only those files with the date you specify. If you do not supply a date, DIR uses the system's current date.
/E	9.2.5	Lists the device directory including unused spaces and their sizes. An empty space on a cassette directory represents a deleted file.
/F	9.2.6	Prints in five columns a short directory (file names and types only) of the device you specify.
/G	9.2.7	Lists the file you specify and all files that follow it in the directory. This option does not list any files that precede the file you specify.
/J[:date]	9.2.8	Prints a directory of the files created on or after the date you specify. If you do not supply a date, DIR uses the system's current date.
/K[:date]	9.2.9	Prints a directory of files created before the date you specify. If you do not supply a date, DIR uses the system's current date.
/L	9.2.10	Lists the directory of the device you specify, including the number of files, their dates, and the number of blocks each file occupies. (This is the default operation.)
/M	9.2.11	Lists a directory of unused areas of the device you specify.
/N	9.2.12	Lists a summary of the device directory.
/O	9.2.13	Similar to /L but lists the sizes and block numbers of the files in octal.
/P	9.2.14	Prints a directory of the device you specify, excluding the files you list.
/Q	9.2.15	Lists a directory of the device you specify, listing the file names and types, sizes, creation dates and starting block numbers of files that have been deleted and whose file name information has not been destroyed.
/R	9.2.16	Lists the files in the reverse order of the sort specified with /A or /S.
/S[:xxx]	9.2.17	Lists the directory of the device you specify in the order you specify; xxx indicates the order in which DIR sorts the listing (xxx can be DAT, NAM, POS, SIZ, or TYP).

9.2.1 The Alphabetical Option (/A)

The /A option lists the directory of the device you specify in alphabetical order by file name and type. It has the same effect as the /S:NAM option. The following example lists the directory of device DX0: in alphabetical order on the terminal.

```
*DX0:/A
 13-Apr-77
DIR    .SAV     16 15-Mar-77     LP     .SYS      2 01-Mar-77
DT     .SYS      2 01-Mar-77     MACRO  .SAV     46 01-Mar-77
DUP    .SAV     17 04-Mar-77     PIP    .SAV     16 16-Mar-77
DXMNSJ.SYS      91 01-Mar-77     SYSMAC.MAC     27 18-Feb-77
EDIT   .SAV     21 01-Mar-77     TT     .SYS      2 01-Mar-77
LINK   .SAV     25 01-Mar-77
 11 Files, 265 Blocks
 215 Free Blocks
```

9.2.2 The Block Number Option (/B)

The /B option prints a directory of the device you specify and includes the starting block number in decimal of all the files listed. The following example lists the directory of device DX0:, including the starting block numbers of files.

```
*DX0:/B
 13-Apr-77
DXMNSJ.SYS      91 01-Mar-77      14     TT     .SYS      2 01-Mar-77     105
LP     .SYS      2 01-Mar-77     107     DT     .SYS      2 01-Mar-77     109
EDIT   .SAV     21 01-Mar-77     111     LINK   .SAV     25 01-Mar-77     132
DUP    .SAV     17 04-Mar-77     157     DIR    .SAV     16 15-Mar-77     174
PIP    .SAV     16 16-Mar-77     190     MACRO  .SAV     46 01-Mar-77     206
SYSMAC.MAC      27 18-Feb-77     252
 11 Files, 265 Blocks
 215 Free blocks
```

9.2.3 The Columns Option (/C:n)

The /C[:n] option lists the directory in the number of columns you specify. The argument, n, represents an integer in the range 1-9. If you do not use the /C:n option, DIR lists the directory in two columns for normal listings and five columns for abbreviated listings. The following command, for example, lists on the terminal the directory of device DX1: in one column.

```
*DX1:/C:1
 13-Apr-77
FORTRA.SAV    191 28-Feb-77
BASIC .SAV     51 25-Feb-77
SYSLIB.OBJ    200 31-Mar-77
 2 Files, 442 Blocks
 38 Free blocks
```

9.2.4 The Date Option (/D[:date])

The /D[:date] option includes in the directory listing only those files with the date you specify. The default date is the system's current date. For example, the following command lists on the terminal all the files that were created on 1 March 1977.

```
*DX0:/D:01,:MAR:77,
 13-Apr-77
DXMNSJ.SYS     91 01-Mar-77     TT     .SYS      2 01-Mar-77
LP     .SYS     2 01-Mar-77     DT     .SYS      2 01-Mar-77
EDIT   .SAV    21 01-Mar-77     LINK   .SAV     25 01-Mar-77
MACRO  .SAV    46 01-Mar-77
 7 Files, 149 Blocks
 215 Free blocks
```

9.2.5 The Entire Option (/E)

The /E option lists the entire directory including the unused areas and their sizes in blocks (decimal). The following example lists on the terminal the entire directory of device DX1:, including unused areas.

```
*DX1:/E
 03-May-77
DIR    .SAV    16 08-Apr-77     DUP    .SAV     17 13-Apr-77
ABC    .MAC     4 19-Apr-77     AAF    .MAC      2 19-Apr-77
PIP    .SAV    16 14-Apr-77     COMB   .SAV      4 19-Apr-77
MERGE  .FOR     6 24-Apr-77     < UNUSED >     415
 7 Files, 65 Blocks
 415 Free blocks
```

9.2.6 The Fast Option (/F)

The /F option lists only file names and file types, omitting file lengths and associated dates. For example, the following command lists on the terminal only file names and types from device DT0:.

```
*DT0:/F
 13-Apr-77
DMPX   .MAC     MATCH  .BAS     EXAMP  .FOR     GRAPH  .FOR     SPOOL  .MAC
GLOBAL.MAC      PROSEC.MAC      KB     .MAC     EXAMP  .MAC     FIX463.SAV
GRAPH  .BAK     DTMNSJ.SYS
 12 Files, 167 Blocks
 397 Free blocks
```

9.2.7 The Begin Option (/G)

The /G option lists the directory of the device you specify, beginning with the file you specify and including all the files that follow it in the directory. Usually, the disk you are using as a system device contains a number of files that the operating system needs. These files include .SYS monitor files, .SAV utility program files, and various .OBJ, .MAC, and .BAT files. They are generally grouped together and usually list at the beginning of a normal device directory. Files that you create and use, such as source files and text files, are also grouped together and follow the operating system files in the directory. If you specify the name of the last system file with the /G in the command line, DIR prints a directory of only those files that you created and stored on the device. The following command, for example, lists the last system file (CT.SYS) and all the user files that follow it.

```
*DX0:CT.SYS/G
 15-Apr-77
CT     .SYS     5 08-Apr-77     TEXT   .TST     18 28-Jan-77
PROG   .BAS     3 15-Apr-77     DMPX   .MAC      3 15-Apr-77
MATCH  .BAS     3 15-Apr-77     EXAMP  .FOR      2 15-Apr-77
GRAPH  .FOR     2 15-Apr-77     GLOBAL.MAC      2 15-Apr-77
PROSEC.MAC      2 15-Apr-77     KB     .MAC     33 15-Apr-77
EXAMP  .MAC     4 15-Apr-77     FIX463.SAV      2 29-Jul-76
GRAPH  .BAK    18 28-Jan-77
 13 Files, 97 Blocks
 199 Free blocks
```

9.2.8 The Since Option (J[:date])

The /J[:date] option lists a directory of all files stored on the device you specify that were created on or after the date you supply. The default date is the system's current date. The following command lists on the terminal all files on device DT0: that were created on or after 28 January 77.

```
*DT0:/J:28.:JAN:77.
 13-Apr-77
GRAPH .BAK     18 28-Jan-77     DTMNSJ.SYS     91 01-Mar-77
 2 Files, 109 Blocks
 397 Free blocks
```

9.2.9 The Before Option (/K[:date])

The /K[:date] option prints a directory of files created before the date you specify. The default date is the system's current date. The following command lists on the terminal all files stored on device DX1: that were created before 15 March 1977.

```
*DX1:/K:15.:MAR:77.
 13-Apr-77
FORTRA.SAV     191 28-Feb-77    BASIC .SAV     51 25-Feb-77
 2 Files, 242 Blocks
 38 Free blocks
```

9.2.10 The Listing Option (/L)

The /L option lists the directory of the device you specify. The listing contains the current date, all files and their associated creation dates, the number of blocks used by each file, total free blocks on the device (if disk or DECtape), the number of files listed, and the total number of blocks used by the files. File lengths, number of blocks and number of files are indicated as decimal values. For example, the following command lists on the line printer the directory for device DT1:.

```
*LP:=DX1:/L
```

The line printer output looks like this:

```
 03-MAY-77
DIR    .SAV     16 08-APR-77     DUP    .SAV     17 13-APR-77
ABC    .MAC      4 19-APR-77     AAF    .MAC      2 19-APR-77
PIP    .SAV     16 14-APR-77     MERGF  .FOR      6 24-APR-77
 6 FILES, 61 BLOCKS
 419 FREE BLOCKS
```

9.2.11 The Unused Areas Option (/M)

The /M option prints only a directory of unused areas and their size on the device you specify. For example, the following command lists on the terminal all the unused areas on device DK:.

```
*/M
 03-May-77
< UNUSED >     21               < UNUSED >     295
< UNUSED >     16               < UNUSED >     594
 0 Files, 0 Blocks
 926 Free blocks
```

9.2.12 The Summary Option (/N)

The /N option prints a summary of the device directory. The following command lists on the terminal the summary of the directory for device DK:.

```
*/N
 13-Apr-77

    72 Files in segment 1

    72 Files in segment 2

    72 Files in segment 3

    12 Files in segment 4

    16 Available segments, 4 in use

 228 Files, 4141 Blocks
 621 Free blocks
```

9.2.13 The Octal Option (/O)

The /O option is similar to the /L option, but lists the sizes and starting block numbers (if you use /B) of the files in octal. If the device you specify is a magnetic tape or cassette, DIR prints the sequence number in octal. For example, the following command lists on the terminal the directory of device DX0:, with sizes in octal.

```
*DX0:/O
 13-Apr-77 Octal
DXMNSJ.SYS     133 01-Mar-77     TT     .SYS       2 01-Mar-77
LP     .SYS      2 01-Mar-77     DT     .SYS       2 01-Mar-77
EDIT   .SAV     25 01-Mar-77     LINK   .SAV      31 01-Mar-77
DUP    .SAV     21 04-Mar-77     DIR    .SAV      20 15-Mar-77
PIP    .SAV     20 16-Mar-77     MACRO  .SAV      56 01-Mar-77
SYSMAC.MAC      33 18-Feb-77
 11 Files, 411 Blocks
 327 Free blocks
```

9.2.14 The Exclude Option (/P)

The /P option lists a directory of all files on a specific device, excluding those that you list. You can specify up to six file specifications.

```
*DX1:*.SAV/P
 03-May-77
ABC    .MAC     4 19-Apr-77      AAF    .MAC      2 19-Apr-77
MERGE  .FOR     6 24-Apr-77
 3 Files, 12 Blocks
 419 Free blocks
```

This command lists on the terminal all files on device DX1: except .SAV files.

9.2.15 The Deleted Option (/Q)

The /Q option lists a directory of the device you specify, listing the file names, types, sizes, creation dates, and starting block numbers in decimal of files that have been deleted but whose file name information has not been destroyed. The file names that print represent either tentative files or files that have been deleted. This can be

useful in recovering files that have been accidentally deleted. Once you identify the file name and location, you can use DUP to rename the area. See Section 8.2.1 for this procedure.

```
*DISK.DIR=/Q
```

This command creates a file called DISK.DIR on device DK: that contains directory information about unused areas from device DK:.

Use the monitor TYPE command to read the file:

```
.TYPE DISK.DIR/LOG
 Files copied:
DK:DISK.DIR     to TT:
 03-May-77
EDIT    DEM      21 03-May-77    3843  DUM    .       295 03-May-77   3882
DEMOF1.OBJ       16 26-Apr-77    4179  DISK   .DIR    297 03-May-77   4206
SCOPE .PIC      297 03-May-77    4503
 0 Files, 0 Blocks
 0 Free blocks
```

9.2.16 The Reverse Option (/R)

The /R option lists a directory in the reverse order of the sort you specify with the /A or /S option.

```
*DX0:/S:DAT/R
 13-Apr-77
PIP    .SAV     16 16-Mar-77     LINK   .SAV     25 01-Mar-77
DIR    .SAV     16 15-Mar-77     LP     .SYS      2 01-Mar-77
DUP    .SAV     17 04-Mar-77     MACRO  .SAV     46 01-Mar-77
DT     .SYS      2 01-Mar-77     TT     .SYS      2 01-Mar-77
DXMNSJ.SYS     91 01-Mar-77     SYSMAC.MAC     27 18-Feb-77
EDIT   .SAV     21 01-Mar-77
 11 Files, 265 Blocks
 215 Free blocks
```

This command lists on the terminal the directory of device DX0: in reverse chronological order.

9.2.17 The Sort Option (/S[:xxx])

The /S[:xxx] option sorts the directory of the specified device according to a 3-character code you specify with :xxx. Table 9-2 summarizes the codes and their functions.

Table 9-2 Sort Codes

Code	Explanation
DAT	Sorts the directory chronologically by creation date. Files that have the same date are sorted alphabetically by file name and file type.
NAM	Sorts the directory alphabetically by file name. Files that have the same file name are sorted alphabetically by file type (this has the same effect as the /A option).
POS	Lists the files in order by their position on the device. This is the same as using /S with no code.
SIZ	Sorts the directory based on file size in blocks. Files that are the same size are sorted alphabetically by file name and file type.
TYP	Sorts the directory alphabetically by file type. Files that have the same file type are sorted alphabetically by file name.

The following examples illustrate the /S option.

```
*DX0:/S:DAT
 13-Apr-77
SYSMAC.MAC     27 18-Feb-77     MACRO .SAV     46 01-Mar-77
DT    .SYS      2 01-Mar-77     TT    .SYS      2 01-Mar-77
DXMNSJ.SYS     91 01-Mar-77     DUP   .SAV     17 04-Mar-77
EDIT  .SAV     21 01-Mar-77     DIR   .SAV     16 15-Mar-77
LINK  .SAV     25 01-Mar-77     PIP   .SAV     16 16-Mar-77
LP    .SYS      2 01-Mar-77
 11 Files, 265 Blocks
 215 Free blocks

*DX0:/S:NAM
13-Apr-77
DIR   .SAV     16 15-Mar-77     LP    .SYS      2 01-Mar-77
DT    .SYS      2 01-Mar-77     MACRO .SAV     46 01-Mar-77
DUP   .SAV     17 04-Mar-77     PIP   .SAV     16 16-Mar-77
DXMNSJ.SYS     91 01-Mar-77     SYSMAC.MAC     27 18-Feb-77
EDIT  .SAV     21 01-Mar-77     TT    .SYS      2 01-Mar-77
LINK  .SAV     25 01-Mar-77
 11 Files, 265 Blocks
 215 Free blocks

*DX0:/S:POS
 13-Apr-77
DXMNSJ.SYS     91 01-Mar-77     DUP   .SAV     17 04-Mar-77
TT    .SYS      2 01-Mar-77     DIR   .SAV     16 15-Mar-77
LP    .SYS      2 01-Mar-77     PIP   .SAV     16 16-Mar-77
DT    .SYS      2 01-Mar-77     MACRO .SAV     46 01-Mar-77
EDIT  .SAV     21 01-Mar-77     SYSMAC.MAC     27 18-Feb-77
LINK  .SAV     25 01-Mar-77
 11 Files, 265 Blocks
 215 Free blocks

*DX0:/S:TYP
 13-Apr-77
SYSMAC.MAC     27 18-Feb-77     PIP   .SAV     16 16-Mar-77
DIR   .SAV     16 15-Mar-77     DT    .SYS      2 01-Mar-77
DUP   .SAV     17 04-Mar-77     DXMNSJ.SYS     91 01-Mar-77
EDIT  .SAV     21 01-Mar-77     LP    .SYS      2 01-Mar-77
LINK  .SAV     25 01-Mar-77     TT    .SYS      2 01-Mar-77
MACRO .SAV     46 01-Mar-77
 11 Files, 265 Blocks
 215 Free blocks

*DX0:/S:SIZ
 13-Apr-77
DT    .SYS      2 01-Mar-77     EDIT  .SAV     21 01-Mar-77
LP    .SYS      2 01-Mar-77     LINK  .SAV     25 01-Mar-77
TT    .SYS      2 01-Mar-77     SYSMAC.MAC     27 18-Feb-77
DIR   .SAV     16 15-Mar-77     MACRO .SAV     46 01-Mar-77
PIP   .SAV     16 16-Mar-77     DXMNSJ.SYS     91 01-Mar-77
DUP   .SAV     17 04-Mar-77
 11 Files, 265 Blocks
 215 Free blocks
```

CHAPTER 10
MACRO-11 PROGRAM ASSEMBLY

This chapter describes how to assemble MACRO-11 programs under the RT-11 operating system, assuming that you have written those programs according to the rules stated in the *PDP-11 MACRO-11 Language Reference Manual*, used associated debugging tools and the linker (see Chapter 11), and understand the RT-11 operating system.

The MACRO-11 assembler operates in two distinct phases, or passes. Chapter 1 of the *PDP-11 MACRO-11 Language Reference Manual* contains a detailed description of the two-pass assembler action.

The assembly output includes any or all of the following items:

1. A binary object file – the machine-readable logical equivalent of the MACRO-11 assembly language source code
2. A listing of the source input file
3. A cross-reference file listing
4. A table of contents listing
5. A symbol table listing

To use the MACRO-11 assembler correctly under RT-11 control, you should understand how to:

1. Initiate and terminate the MACRO-11 assembler (including how to format command strings to specify files MACRO-11 uses during assembly)
2. Assign temporary work files to non-default devices, if necessary
3. Use file specification options to override file control directives in the source program
4. Use the small version of MACRO-11 for PDP-11 systems with 8K memory, if necessary
5. Interpret error messages

The following sections describe these topics.

10.1 INITIATING THE MACRO-11 ASSEMBLER

To call the MACRO-11 assembler from the system device, respond to the system prompt (a dot printed by the keyboard monitor) by typing:

```
R MACRO (RET)
```

When the assembler responds with an asterisk (*), it is ready to accept command string input. (You can also call the assembler using the keyboard monitor MACRO command; see Chapter 4 for a description of this command.)

The assembler now expects a command string consisting of the following items, in sequence

1. Output file specifications
2. An equal sign
3. Input file specifications

Format this command string as follows (punctuation is required where shown):

```
dev:obj,dev:list,dev:cref/s:arg=dev:sourcei, . . . ,dev:sourcen/s:arg (RET)
```

where

dev	is any legal RT-11 device for output; any file-structured device for input
obj	is the file specification of the binary object file that the assembly process produces; the dev for this file should not be TT or LP
list	is the file specification of the assembly and symbol listing that the assembly process produces
cref	is the file specification of the CREF temporary cross-reference file that the assembly process produces. (Omission of device:cref does not preclude a cross-reference listing, however.)
/s:arg	is a set of file specification options and arguments. Section 10.2 describes these options and associated arguments. Before that section, they are omitted from examples.
sourcei	Each sourcei is a file specification for an ASCII MACRO-11 source file or MACRO library file. These files contain the MACRO language programs that you need to assemble. You can specify as many as six source files.

The following command string calls for an assembly that uses one source file plus the system MACRO library to produce an object file BINF.OBJ and a listing. The listing goes directly to the line printer.

```
*DK:BINF.OBJ,LP:=DK:SRC.MAC
```

All output file specifications are optional. The system does not produce an output file unless the command string contains a specification for that file.

The system determines the file type of an output file specification by its position in the command string, as determined by the number of commas in the string. For example, to produce only a listing, and no object file, you must include an empty object specification.

To omit the object file, you must begin the command string with a comma. The following command produces a listing, including cross-reference tables, but not binary object files.

```
*,LP:/C=(source file specification)
```

Notice that you need not include a comma after the final output file specification in the command string.

Table 10-1 lists the default values for each file specification.

10.2 TERMINATING THE MACRO-11 ASSEMBLER

If you have typed R MACRO and received the asterisk prompt but have not yet entered the command string, you can terminate MACRO-11 control by typing CTRL/C once. After you have completed the command string (thus beginning an assembly) you can halt the assembly process at any time by typing CTRL/C twice. This returns control to the system monitor, and a system monitor dot prompt appears on the terminal.

To restart the assembly process, type R MACRO in response to the system monitor prompt. You can also restart using the REENTER command in most cases; however, the RT-11 system does not accept the REENTER command if the assembler is producing a cross-reference listing when you halt the assembly.

Table 10-1 Default File Specification Values

File	Default Device	Default File Name	Default File Type
Object	DK:	Must specify	.OBJ
Listing	Same as for object file	Must specify	.LST
Cref	DK:	Must specify	.TMP
First source	DK:	Must specify	.MAC
Additional source	Same as for preceding source file	Must specify	.MAC
System MACRO Library	System device SY:	SYSMAC	.SML
User MACRO Library	DK: if first file, otherwise same as for preceding source file	Must specify	.MAC

10.3 TEMPORARY WORK FILE

Some assemblies need more symbol table space than available memory can contain. When this occurs the system automatically creates a temporary work file called WRK.TMP to provide extended symbol table space.

The default device for WRK.TMP is DK. To cause the system to assign a different device, enter the following command:

```
.ASSIGN dev: WF
```

The dev parameter is the logical name of a file-structured device. The system assigns WRK.TMP to this device.

10.4 FILE SPECIFICATION OPTIONS

At assembly time you may need to override certain MACRO directives appearing in the source programs. You may also need to direct MACRO-11 on the handling of certain files during assembly. You can satisfy these needs by including special options in the MACRO-11 command string in addition to the file specifications. Table 10-2 lists the options and describes generally the effect of each.

The general format of the MACRO-11 command string is repeated below for your convenience:

```
dev:obj,dev:list,dev:cref/s:arg=dev:source1, . . . ,dev:sourcen/s:arg
```

Table 10-2 File Specification Options

Option	Usage
/L:arg	Listing control, overrides source program directive .LIST
/N:arg	Listing control, overrides source program directive .NLIST
/E:arg	Object file function enabling, overrides source program directive .ENABL
/D:arg	Object file function disabling, overrides source program directive .DSABL
/M	Indicates input file is MACRO library file
/C:arg	Control contents of cross-reference listing
/P:arg	Specifies whether input source file is to be assembled during pass 1 or pass 2

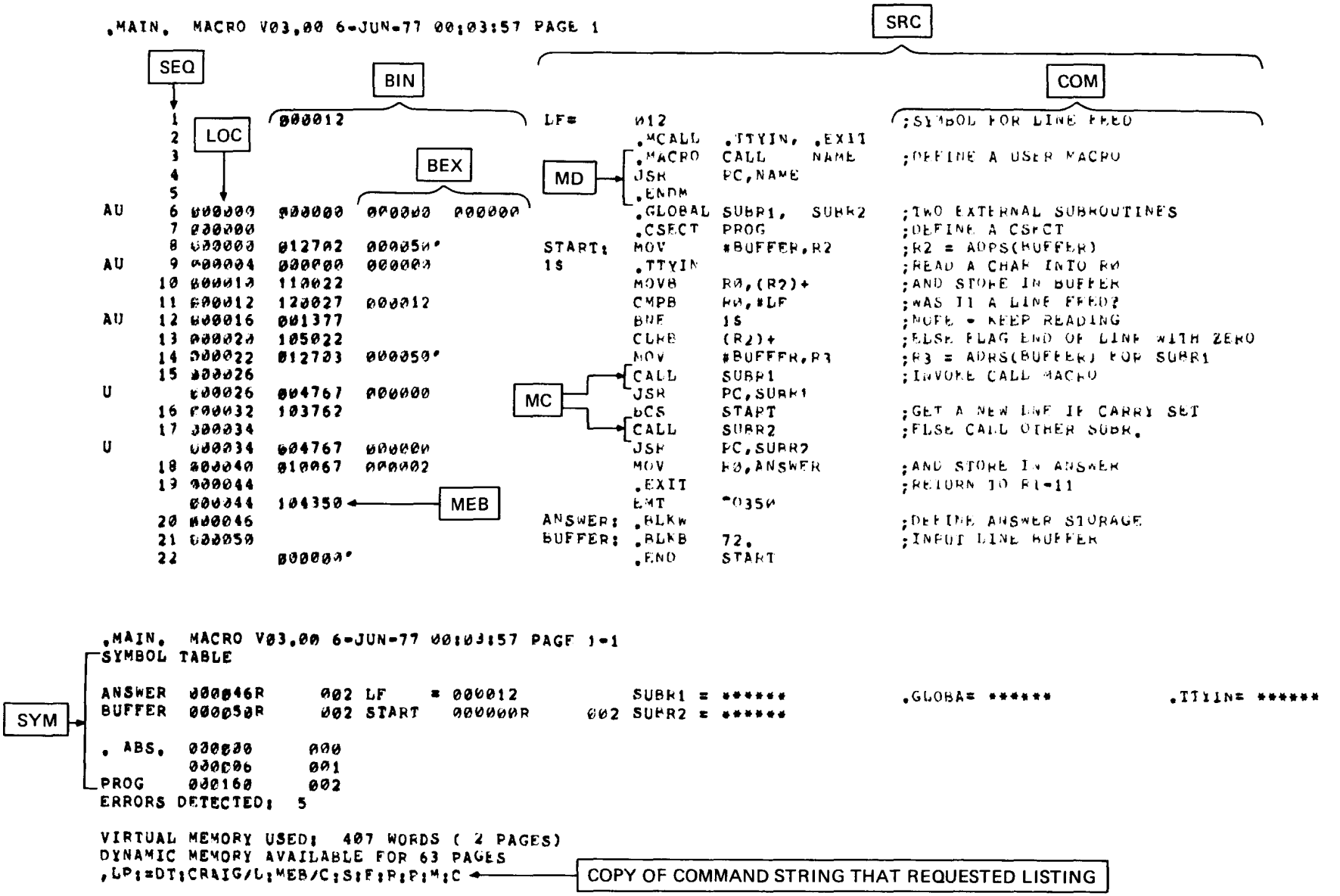


Figure 10-1 Sample Assembly Listing

The /M and /P options affect only the particular source file specification to which they are directly appended in the command string.

Other options are unaffected by their placement in the command string. The /L option, for example, affects the listing file, regardless of where you place it in the command string.

The following subsections describe in detail how to use the several file specification options.

10.4.1 Listing Control Options

Two options, /L:arg and /N:arg, pertain to listing control. By specifying these options with a set of selected arguments (see Table 10-3) you can control the content and format of assembly listings. You can override at assembly time the arguments of .LIST and .NLIST directives in the source program.

Figure 10-1 shows an assembly listing of a small program. This listing shows the more important listing features. It labels each feature with the mnemonic ASCII argument that determines its appearance on the listing; the argument SEQ, for instance, controls the appearance of the source line sequence numbers.

Specifying the /N option with no argument causes the system to list only the symbol table, the table of contents, and error messages.

Specifying the /L option with no arguments causes the system to ignore .LIST and .NLIST directives that have no arguments.

The following example lists binary code throughout the assembly using the 132-column line printer format, and suppresses the symbol table listing.

```
*I,LP:/L:MEB/N:SYM=FILE
```

Table 10-3 Valid Arguments for /L and /N Options

Argument	Default	Controls Listing of
SEQ	list	Source line sequence number
LOC	list	Address location counter
BIN	list	Generated binary code
BEX	list	Binary extensions
SRC	list	Source code
COM	list	Comment
MD	list	Macro definitions, repeat range expansion
MC	list	Macro calls, repeat range expansion
ME	no list	Macro expansions
MEB	no list	Macro expansion binary code
CND	list	Unsatisfied conditionals, .IF and .ENDC statements
LD	no list	List control directives with no arguments
TOC	list	Table of Contents
TTM	no list	132-column line printer format when not specified, terminal mode when specified
SYM	list	Symbol table

10.4.2 Function Control Options

Two options, /E:arg and /D:arg allow you to enable or disable functions at assembly time, and thus influence the form and content of the binary object file. These functions can override ENABL and DSABL directives in the source program.

Table 10-4 summarizes the acceptable /E and /D function arguments, their normal default status, and the functions they control. See Section 5.5.2 for further details of the functions.

Table 10-4 Valid Arguments for /E and /D Options

Argument	Default Mode	Function
ABS	Disable	Allows absolute binary output
AMA	Disable	Assembles all absolute addresses as relative addresses
CDR	Disable	Treats all source information beyond column 72 as commentary
CRF	Enable	Allows cross-reference listing. Disabling this function inhibits CREF output if option /C is active
FPT	Disable	Truncates floating point values (instead of rounding)
GBL	Disable	Treats undefined symbols as globals
LC	Disable	Allows lower case ASCII source input
LSB	Disable	Allows local symbol block
PNC	Enable	Allows binary output
REG	Enable	Allows mnemonic definitions of registers

For example, if you type the following commands the system assembles a file while treating columns 73 through 80 of each source card as commentary.

```
.R PIP
*CARDS.MAC=CR:/A
*^C
.R MACRO
*,LP:=CARDS.MAC/E:CDR
```

Because MACRO-11 is a two-pass assembler, you cannot read the cards directly from the card reader or other non-file structured device. You must use PIP (or the keyboard monitor COPY command) to transfer input to a file-structured device before beginning the assembly.

Use either the function control or listing control option and arguments at assembly time to override corresponding listing or function control directives in the source program. For example, assume that the source program contains the following sequence:

```
.NLIST MEB
   .


.LIST MEB
```

In this example, you disable the listing of macro expansion binary code for some portion of the code and subsequently resume MEB listing. However, if you indicate /L:MEB in the assembly command string, the system ignores both the .NLIST MEB and the .LIST MEB directives. This enables MEB listing throughout the program.

10.4.3 Macro Library File Designation Option

The /M option is meaningful only if appended to a source file specification. It has no arguments, and it designates its associated source file as a macro library.

If the command string does not include the standard system macro library SYSMAC.SML, the system automatically includes it as the last source file in the command string.

When the assembler encounters an .MCALL directive in the source code, it searches macro libraries according to their order of appearance in the command string. When it locates a macro record whose name matches that given in the .MCALL, it assembles the macro as indicated by that definition. Thus if two or more macro libraries contain definitions of the same macro name, the macro library that appears leftmost in the command string takes precedence.

Consider the following command string:

```
*(output file specification)=ALIB.MAC/M,BLIB.MAC/M,XIZ
```

Assume that each of the two macro libraries, ALIB and BLIB, contain a macro called .BIG, but with different definitions. Then, if source file XIZ contains a macro call .MCALL .BIG, the system includes the definition of .BIG in the program as it appears in the macro library ALIB.

Moreover, if macro library ALIB contains a definition of a macro called .READ, that definition of .READ overrides the standard .READ macro definition in SYSMAC.SML.

10.4.4 Cross-Reference (CREF) Table Generation Option

A cross-reference (CREF) table lists all or a subset of the symbols in a source program, identifying the statements that define and use symbols.

10.4.4.1 Obtaining a Cross-Reference Table – To obtain a CREF table you must include the /C:arg option in the command string. Usually you include the /C:arg option with the assembly listing file specification. You can in fact place it anywhere in the command string.

If the command string does not include a cref file specification, the system automatically generates a temporary file on device DK:. If you need to have a device other than DK: contain the temporary cref file, you must include the dev:cref field in the command string.

If the listing device is magtape or cassette, load the handler for that device before issuing the command string, using the monitor LOAD command (described in Chapter 4).

A complete CREF listing contains the following six sections:

1. A cross reference of program symbols; that is, labels used in the program and symbols followed by a – operator.
2. A cross reference of register equate symbols; that is, symbols defined in the program by the construct:

   symbol–n

   Normally, these symbols include R0, R1, R2, R3, R4, R5, SP, and PC.

3. A cross reference of MACRO symbols; that is, those symbols defined by .MACRO and .MCALL directives.
4. A cross reference of permanent symbols, that is, all operation mnemonics and assembler directives.
5. A cross reference of program sections. These symbols include the names you specify as operands of .CSECT or .PSECT directives.
6. A cross reference of errors. The system groups and lists all flagged errors from the assembly by error type.

You can include any or all of these six sections on the cross-reference listing by specifying the appropriate arguments with the /C option. These arguments are listed and described in Table 10-5.

Table 10-5 /C Option Arguments

Argument	CREF Section
S	User defined symbols
R	Register symbols
M	MACRO symbolic names
P	Permanent symbols including instructions and directives
C	Control and program sections
E	Error code grouping

NOTE

Specifying /C with no arguments is equivalent to specifying /C:S:M:E. That special case excepted, you must explicitly request each CREF section by including its arguments. No cross-reference file occurs if the /C option is not specified, even if the command string includes a CREF file specification.

10.4.4.2 Handling Cross-Reference Table Files – When you request a cross-reference listing by means of the /C option, you cause the system to generate a temporary file, DK:CREF.TMP.

If device DK: is write-locked or if it contains insufficient free space for the temporary file, you can allocate another device for the file. To allocate another device, specify a third output file in the command string; that is, include a dev:cref specification. (You must still include the /C option to control the form and content of the listing. The dev:cref specification is ignored if the /C option is not also present in the command string.)

The system then uses the dev:cref file instead of DK:CREF.TMP and deletes it automatically after producing the CREF listing.

The following command string causes the system to use RK2:TEMP.TMP as the temporary CREF file.

```
*,LP:/,RK2:TEMP.TMP=SOURCE/C
```

Another way to assign an alternative device for the CREF.TMP file is to enter the following command prior to entering R MACRO:

```
.ASSIGN dev:CF
```

This method is preferred if you intend to do several assemblies, as it relieves you from having to include the dev:cref specification in each command string. If you enter the ASSIGN dev: CF command, and later include a cref specification in a command string, the specification in the command string prevails for that assembly only.

The system lists requested cross-reference tables following the MACRO assembly listing. Each table begins on a new page. (Figure 10-2 combines the tables to save space, however.)

The system prints symbols and also symbol values, control sections, and error codes, if applicable, beginning at the left margin of the page. References to each symbol are listed on the same line, left-to-right across the page. The system lists references in the form p-l; where p is the page in which the symbol, control section, or error code appears, and 1 is the line number on the page.

A number sign (#) next to a reference indicates a symbol definition. An asterisk (*) next to a reference indicates a destructive reference – that is, an operation that alters the contents of the addressed location.

10.4.5 Assembly Pass Option

The /P:arg option is meaningful only if appended to a source input file specification. You must specify either of two arguments with it: 1 or 2.

The specification /P:1 calls for assembly of the file during pass 1 only. Some files consist entirely of code that is completely assembled at the end of pass 1. By specifying /P:1 for these files, you can cause MACRO-11 to skip processing of these files through pass 2. In some cases this procedure can save considerable assembly time.

The specification /P:2 calls for assembly of the file during pass 2 only. (NOTE: Situations where the /P:2 option can be meaningfully employed are unusual.)

10.5 MACRO-11 8K VERSION

A subset version of MACRO-11, with file name MAC8K.SAV, is available for systems with 8K words of memory – that is, systems with insufficient memory to support operation of the full MACRO-11 assembler.

The full assembler (MACRO) requires approximately 10K words of memory, or must be operating on at least a 12K system using the single-job (SJ) monitor.

The subset version (MAC8K) requires approximately 6K words of memory, or must be operating on an 8K system using the baseline SJ monitor.

The subset version differs from the full assembler as follows:

1. All handlers must be resident (that is, loaded) before you call MAC8K.
2. The full assembler prints the input command string at the end of the listing; the subset version does not.
3. The subset version does not recognize the following items:
   a. The operation codes exclusive to PDP-11/45 and PDP-11/70
   b. The Commercial Instruction Set (CIS)
   c. The FLT2 and FLT4 floating point directives
4. The system device is the only available file medium under MAC8K.
5. The subset version does not support the cross-reference file and ignores attempts to obtain such a listing.
6. Assembly times of the subset version are noticeably longer.
7. The subset version operates only under control of the baseline single-job monitor (see the *RT-11 System Generation Manual*).

10.6 MACRO-11 ERROR CODES

The MACRO-11 system prints diagnostic error codes as the first character of a source line on which the assembler detects an error. This error code identifies the type of error; for example, a code of M indicates a multiple definition of a label. Table 10-6 shows the error codes that might appear on an assembly listing. For detailed information on

```
.MAIN.  MACRO V03.00 6-JUN-77 00:03:57 PAGE S-1
CROSS REFERENCE TABLE (CREF V01-05 )


.GLOBA    1-6
.TTYIN    1-9
ANSWER    1-19*     1-20*
BUFFER    1-8       1-14      1-21*
LF        1-11*     1-11
START     1-9*      1-16      1-22
SUBR1     1-6       1-15
SUBR2     1-6       1-17



.MAIN.  MACRO V03.00 6-JUN-77 00:03:57 PAGE R-1
CROSS REFERENCE TABLE (CREF V01-05 )

PC        1-15*     1-17*
R0        1-12      1-11      1-18
R2        1-8*      1-13*     1-13*
R3        1-14*



.MAIN.  MACRO V03.00 6-JUN-77 00:03:57 PAGE M-1
CROSS REFERENCE TABLE (CREF V01-05 )

.EXIT     1-2*      1-19
.TTYIN    1-2*
CALL      1-3*      1-15      1-17



.MAIN.  MACRO V03.00 6-JUN-77 00:03:57 PAGE P-1
CROSS REFERENCE TABLE (CREF V01-05)

.BLKB     1-21
.BLKW     1-20
.CSECT    1-7
.END      1-22
.MACRO    1-3
.MCALL    1-2
BCS       1-16
BNE       1-12
CLRB      1-13
CMPB      1-11
EMT       1-19
JSR       1-15      1-17
MOV       1-8       1-14      1-18
MOVB      1-10



.MAIN.  MACRO V03.00 6-JUN-77 00:03:57 PAGE C-1
CROSS REFERENCE TABLE (CREF V01-05 )

          0-0
. ABS.    0-0
PROG      1-7



.MAIN.  MACRO V03.00 6-JUN-77 00:03:57 PAGE F-1


A         1-6       1-9       1-12
U         1-6       1-9       1-12      1-15      1-17
```

Figure 10-2 Cross-Reference Table

Table 10-6 MACRO-11 Error Codes

Error Code	Meaning
A	Addressing or relocation error. This occurs when an instruction operand has an invalid address, or when the definition of a local symbol occurs more than 128 words from the beginning of a local symbol block.
B	Boundary error. The current setting of the location counter would cause the assembly of instruction or word data at an odd memory address. The system increments the location counter by 1 to correct this.
D	Reference to multiple-definition symbol. The program refers to a non-local label that is defined more than once.
E	No END directive. The assembler has reached the end of a source file and found no END directive. The system generates .END and continues.
I	Illegal character detected. The assembler has encountered in the source file a character that is not included in the language character set. The system replaces each illegal character with a ? on the assembly listing and proceeds as if the illegal character were not present.
L	Link buffer overflow. The assembler has encountered an input line greater than 132 characters. In terminal mode the system ignores additional characters.
M	Multiple definition of a label. The source program is attempting to define a label equivalent in the first six characters to a label defined previously.
N	Decimal point missing from decimal number. A number containing the digit 8 or 9 lacks a decimal point.
O	Op-code error. A directive appears in an inappropriate context.
P	Phase error. The definition or value of a label differs from one pass to another, or a local symbol occurs more than once in a local symbol block.
Q	Questionable syntax. This can have any of several causes, as follows: 1. There are missing arguments. 2. The instruction scan is not complete. 3. A line feed or form feed does not immediately follow a carriage return.
R	Register-type error. The source program attempts an invalid reference to a register.
T	Truncation error. A number generates more than 16 significant bits, or an expression generates more than 8 significant bits while a .BYTE directive is active.
U	Undefined symbol. A symbol not defined elsewhere in the program appears as a factor in an expression. The assembler assigns the undefined symbol a constant zero value.
Z	Incompatible instruction (warning). The instruction is not defined for all PDP-11 hardware configurations.

CHAPTER 11
LINKER (LINK)

The RT-11 linker (LINK) converts object modules produced by an RT-11 supported language translator into a format suitable for loading and execution. If you have no previous experience with the linker, read Chapter 12 of the *Introduction to RT-11* for an introductory-level description of the linking process. You can separately assemble a main program and each of its subroutines without assigning an absolute load address at assembly time. The linker processes the object modules of the main program and subroutines to:

- Relocate each object module and assign absolute addresses
- Link the modules by correlating global symbols that are defined in one module and referenced in another
- Create the initial control block for the linked program that the GET, R, RUN, and FRUN commands use
- Create an overlay structure if specified and include the necessary run-time overlay handlers and tables
- Search libraries you specify to locate unresolved globals
- Automatically search a default system library to locate any remaining unresolved globals
- Produce a load map showing the layout of the load module
- Produce a symbol definition file.

The RT-11 linker requires two passes over the input modules. During the first pass it constructs the symbol table, including all program section names and global symbols in the input modules. After it processes all non-library files, the linker scans the library files to resolve undefined globals. It links only those modules that are required into the root segment (that part of the program that is never overlaid). During the final pass, the linker reads the object modules, performs most of the functions listed above, and produces a load module (which is in memory image format for background jobs or for jobs that run in the single-job environment, relocatable format for foreground jobs, and formatted binary for use with the Absolute Loader).

The linker runs in a minimal RT-11 system of 8K words of memory; the linker uses any additional memory to facilitate efficient linking and to extend the size of the symbol table. The linker accepts input from any random-access device on the system; there must be at least one random-access device (disk or DECtape) for memory image or relocatable format output.

11.1 CALLING AND USING THE LINKER

To call the RT-11 linker from the system device, respond to the dot printed by the keyboard monitor by typing:

R LINK (RET)

The Command String Interpreter prints an asterisk at the left margin on the console terminal when it is ready to accept a command line. If you enter only a carriage return at this point, the linker prints its current version number. Type two CTRL/Cs to halt the linker at any time (or a single CTRL/C to halt the linker when it is waiting for console terminal input) and return control to the monitor. To restart the linker, type R LINK or REENTER in response to the monitor's dot.

The first command string you enter in response to the linker's prompt has this syntax:

[binout-filespec],[mapout-filespec],[stbout-filespec] = obj-filespec[/option . . .] [, . . . obj-filespec[/option . . .]]

where

binout-filespec	represents the device, name and file type to be assigned to the linker's output load module file.
mapout-filespec	represents the device, file name and file type of the load map output file.
stbout-filespec	represents the device, file name and file type of the symbol definition file.
obj-filespec	represents an object module (that can be a library file) to be linked.
/option	is one of the options from Table 11-2.

In each filespec above, the device should be a random access device, with these exceptions: the output device for the load map file can be any RT-11 device, as can the output device for an .LDA file if you use the /L option. If you do not specify a device, the linker uses default device DK:. Note that the linker load map contains lower case characters. Use the SET LP LC command to enable lower case printing if your printer has lower case characters.

If you do not specify an output file, the linker assumes that you do not desire the associated output. For example, if you do not specify the load module and load map (by using a comma in place of each file specification) the linker prints only error messages, if any occur.

Table 11-1 shows the default values for each specification.

Table 11-1 Linker Defaults

	Device	File Name	File Type
Load Module	DK:	none	SAV, REL(/R), LDA(/L)
Map Output	Same as load module	none	MAP
Symbol Definition Output	DK: or same as previous output device	none	STB
Object Module	DK: or same as previous object module	none	OBJ

If you make a syntax error in a command string, the system prints an error message. You can then type a new command string following the asterisk. Similarly, if you specify a nonexistent file, a warning error occurs; control returns to the Command String Interpreter, an asterisk prints and you can enter a new command string.

11.2 OPTIONS SUMMARY

Table 11-2 lists the options associated with the linker. You must precede the letter representing each option by the slash character. Options must appear on the line indicated if you continue the input on more than one line, but you can position them anywhere on the line. (Section 11.8 provides a more detailed explanation of each option.)

Table 11-2 Linker Options

Option Name	Command Line	Section	Explanation
/B:n	first	11.8.1	Changes the bottom address of a program to n (illegal for foreground links).
/C	any but last	11.8.2	Continues input specification on another command line (you can use /C also with /O; do not use /C with the // option).
/E:n	first	11.8.3	Extends a particular program section to a specific value.
/F	first	11.8.4	Instructs the linker to use the default FORTRAN library, FORLIB.OBJ; this option is provided only for compatibility with previous versions of RT-11.
/H:n	first	11.8.5	Specifies the top (highest) address to be used by the relocatable code in the load module.
/I	first	11.8.6	Extracts the global symbols you specify (and their associated object modules) from the library and links them into the load module.
/K:n	first	11.8.7	Inserts the value you specify (the valid range for n is from 1 to 28) into word 56 of block 0 of the image file; this option is provided only for compatibility with the RSTS operating system.
/L	first	11.8.8	Produces a formatted binary output file (illegal for foreground links).
/M or /M:n	first	11.8.9	Causes the linker to prompt you for a global symbol that represents the stack address, or sets the stack address to the value n.
/O:n	any but the first	11.8.10	Indicates that the program is an overlay structure; n specifies the overlay region to which the module is assigned.
/P:n	first	11.8.11	Changes the default amount of space the linker uses for a library routines list.
/R[:n]	first	11.8.12	Produces output in relocatable format and can indicate stack size for a foreground job.
/S	first	11.8.13	Makes the maximum amount of space in memory available for the linker's symbol table. (You only need to use this option when a particular link stream causes a symbol table overflow.)

(Continued on next page)

Table 11-2 (Cont.) Linker Options

Option Name	Command Line	Section	Explanation
/T or /T:n	first	11.8.14	Causes the linker to prompt you for a global symbol that represents the transfer address, or sets the transfer address to the value n.
/U:n	first	11.8.15	Rounds up the section you specify so that the size of the root segment is a whole number multiple of the value you supply (n must be a power of 2).
/W	first	11.8.16	Directs the linker to produce a wide load map listing.
/X		11.8.17	Does not output the bitmap if the code is below 400; this option is provided only for compatibility with the RSTS operating system.
/Y:n	first	11.8.18	Starts a specific program section on a particular address boundary.
/Z:n	first	11.8.19	Sets unused locations in the load module to the value n.
//	first and last	11.8.20	Allows you to specify command string input on additional lines. Do not use this option with /C.

11.3 MEMORY ALLOCATION

The linker allocates the physical memory and address space that the load module requires. The area of memory that the linker allocates for a load module contains the following elements:

- a system communication area
- hardware vectors
- a stack
- a set of named areas called program sections (p-sections).

Section 11.5.2 describes the system communication area.

The stack is an area that a program can use for temporary storage and subroutine linkage. General register 6, the stack pointer (SP), references the stack.

The system communication area, the hardware vectors, and the stack areas are all part of the load module area called the absolute section. The absolute section is often called the ASECT because it is the assembler directive .ASECT that allows information to be stored there. This section appears in the load map with the name .ABS. and is always the first section in the listing. The absolute section (ASECT) normally ends at address 1000 (octal).

A program section is an area of the load module that contains code and/or data; you can reference it by name. The set of attributes associated with each p-section controls the allocation and placement of the section within the load module.

A p-section is the basic unit of memory for a program. It is composed of the following elements:

- a name by which it can be referenced
- a set of attributes that defines its contents, mode of access, allocation, and placement in memory
- a length that determines how much storage is reserved for the p-section.

You create p-sections by using the COMMON statement in FORTRAN, or the .PSECT (or .CSECT) directive in MACRO. You can use the .PSECT (or .CSECT) directive to attach attributes to the section. Note that the attributes that follow the p-section name are not part of the name; only the name itself distinguishes one p-section from another. You should make sure, then, that p-sections of the same name that you want to link together also have the same attribute list. Do this because the linker uses the first appearance of the .PSECT and its attributes throughout the operation. If the linker encounters p-sections with the same name that have different attributes, it prints a warning message.

The linker collects from the input modules scattered references to a p-section and combines them in a single area of the load module. The attributes, which are listed in Table 11-3, control the way the linker collects and places this unit of storage.

Table 11-3 P-section Attributes

Attribute	Value	Explanation
access-code[1]	RW	Read/Write – data can be read from, and written into, the p-section.
	RO	Read Only – data can be read from, but cannot be written into, the p-section.
type-code	D	Data – the p-section contains data.
	I	Instruction – the p-section contains either instructions, or data and instructions.
scope-code	GBL	Global – the p-section name is recognized across overlay segment boundaries. The linker allocates storage for the p-section from references outside the overlay segment.
	LCL	Local – the p-section name is recognized only within each individual overlay segment. The linker allocates storage for the p-section from references within the overlay segment only.
reloc-code	REL	Relocatable – the base address of the p-section is relocated relative to the virtual base address of the program.
	ABS	Absolute – the base address of the p-section is not relocated. It is always 0.
alloc-code	CON	Concatenate – all allocations to a given p-section name are concatenated. The total allocation is the sum of the individual allocations.
	OVR	Overlay – all allocations to a given p-section name overlay each other. The total allocation is the length of the longest individual allocation.

[1] Not used by the linker.

The scope-code and type-code are meaningful only when you define an overlay structure for the program. In an overlaid program, a global section is known throughout the entire program. Object modules contribute to only one global section of the same name. If two or more segments contribute to a global section, then the linker allocates that global section in the root segment of the program. In contrast to global sections, local sections are only known within a particular program segment. Because of this, several local sections of the same name can appear in different segments. Thus, several object modules contributing to a local section do so only within each segment. An example of a global section is named COMMON in FORTRAN. An example of a local section is the default blank section for each macro routine.

The alloc-code determines the starting address and length of memory allocated by modules that reference a common p-section. If the alloc-code indicates that such a p-section is to be overlaid, the linker places the allocations from each module starting at the same location in memory. It determines the total size from the length of the longest reference to the p-section. The last input module that stores information in a particular location determines which values the linker stores in the indicated locations of the load module. If the alloc-code indicates that a p-section is to be concatenated, the linker places the allocations from the modules one after the other in the load module; it determines the total allocation from the sum of the lengths of the contributions.

The allocation of memory for a p-section always begins on a word boundary. If the p-section has the D (data) and CON (concatenate) attributes, all storage that subsequent modules contribute is appended to the last byte of the previous allocation. This occurs whether or not that byte is on a word boundary. For a p-section with the I (instruction) and CON attributes, however, all storage that subsequent modules contribute begins at the nearest following word boundary.

The .CSECT directive of MACRO is converted internally by both MACRO and the linker to an equivalent .PSECT with fixed attributes. An unnamed CSECT (blank section) is the same as a blank PSECT with the following attributes: RW, I, LCL, REL, and CON.

A named CSECT is equivalent to a named PSECT with these attributes: RW, I, GBL, REL, and OVR. Table 11-4 shows these sections and their attributes.

The names assigned to p-sections are not considered to be global symbols; you cannot reference them as such. For example:

```
MOV     #PNAME,R0
```

This statement, where PNAME is the name of a section, is illegal and generates the Undefined global error message if no global symbol of PNAME exists. A symbol can be the same for both a p-section name and a global symbol. The linker treats them separately.

The linker determines the memory allocation of p-sections by the order of occurrence of the p-sections in the input modules. The absolute section (. ABS.) always comes first, followed by the blank section of the input file (if one exists) and the named section. If there is more than one named section, the named sections appear in the same order in which they occur in the input files. For example, the FORTRAN compiler arranges the p-sections in the main program module so that the USR can swap over pure code in low memory rather than over data required by the function making the USR call.

Table 11-4 Section Attributes

	access-code	type-code	scope-code	reloc-code	alloc-code
CSECT	RW	I	LCL	REL	CON
CSECT name	RW	I	GBL	REL	OVR
ASECT	RW	I	GBL	ABS	OVR
COMMON/name/	RW	D	GBL	REL	OVR

11.4 GLOBAL SYMBOLS

Global symbols provide the link, or communication, between object modules. You create global symbols with the .GLOBL or .ENABL GBL assembler directive (or with double colon, ::, or double equal sign, ==). If the global symbol is defined in an object module (as a label using :: or by direct assignment using ==), other object modules can reference it. If the global symbol is not defined in the object module, it is an external symbol and is assumed to be defined in some other object module. If a global symbol is used as a label in a routine, it is often called an entry point. That is, it is an entry point to that subroutine.

As the linker reads the object modules it keeps track of all global symbol definitions and references. It then modifies the instructions and data that reference the global symbols. The linker always prints undefined globals on the console terminal after pass-1. If you request a load map on the terminal, they appear at the end of the load map.

Table 11-5 shows how the linker resolves global references when it creates the load module.

Table 11-5 Global Reference Resolution

Module Name	Global Definition	Global Reference
IN1	B1 B2	A L1 C1 XXX
IN2	A B1	B2
IN3		B1

In processing the first module, IN1, the linker finds definitions for B1 and B2, and references to A, L1, C1, and XXX. Because no definition currently exists for these references, the linker defers the resolution of these global symbols. In processing the next module, IN2, the linker finds a definition for A that resolves the previous reference, and a reference to B2 that can be immediately resolved.

When all the object modules have been processed, the linker has three unresolved global references remaining: C1, L1, and XXX. A search of the default system library resolves XXX. The global symbols C1 and L1 remain unresolved and are, therefore, listed as undefined global symbols.

The relocatable global symbol, B1, is defined twice and is listed on the terminal as a multiply defined global symbol. The linker uses the first definition of a multiply defined symbol. An absolute global symbol can be defined more than once without being listed as multiply defined as long as each occurrence of the symbol has the same value.

11.5 INPUT AND OUTPUT

Linker input and output is in the form of modules; the linker uses one or more input modules to produce a single output (load) module.

11.5.1 Object Modules

Object files, consisting of one or more object modules, are the input to the linker (the linker ignores files that are not object modules). Object modules are created by an appropriate language translator. The module name item declares the name of the object module. The first six Radix-50 characters of the .TITLE assembler directive are used as the name of the object module. These six characters must be Radix-50 characters (the linker ignores any characters beyond the sixth character). The linker prints the first module name it encounters in the input file stream (normally the main routine of the program) on the second line of the map following .TITLE. It ignores additional module names. The linker reads each object module twice. During the first pass it reads each object

module to construct a symbol table and to assign absolute values to the program section names and global symbols. The linker uses the library files to resolve undefined globals. It places their associated object modules in the root. On the second and final pass, the linker reads the object modules, links and relocates the modules and outputs the load module.

11.5.2 Load Module

The primary output of the linker is a load module that you can run under RT-11. The linker creates as a load module a memory image file (SAV) for use under a single-job system or the background job. If you need to execute a program in the foreground, use the /R option to produce a relocatable format (REL) foreground load module. The linker can produce an absolute load module (LDA) if you need to load the module with the Absolute Loader.

The load module for a memory image file is arranged as follows:

Root Segment	Overlay Segments (optional)

For a relocatable image file the load modules are arranged as follows:

Root Segment	Overlay Segments (optional)	Relocation information for root and overlay segments

The first 256-word block of the root segment (main program) contains the memory usage bit map and the locations the linker uses to pass program control parameters. The memory usage bit map outlines the blocks of memory the load module uses; it is located in locations 360 through 377.

The control parameters are located in locations 40 through 50. They contain the following information when the module is loaded:

Address	Information
40	Start address of program
42	Initial setting of SP (stack pointer)
44	Job status word (overlay bit set by LINK)
46	USR swap address (0 implies normal location)
50	Highest memory address in program (high limit)

The linker stores default values in locations 40, 42, and 50, unless you use options to specify otherwise. The /T option affects location 40, for example, and /M affects location 42. You can also use the .ASECT directive to change the defaults. The overlay bit is located in the job status word. LINK automatically sets this bit if the program is overlaid. Otherwise, the linker initially sets location 44 to 0. Location 46 also contains zero unless you specify another value by using the .ASECT directive.

For a foreground link, the following additional parameters contain information:

Address	Information
14, 16	(XM only) BPT trap
20, 22	(XM only) IOT trap
34, 36	TRAP vector
52	Size of root segment in bytes
54	Stack size in bytes (value with /R or default 128)
56	Size of overlay region in bytes
60	Identification that file is in relocatable (REL) format
62	Relative block number for start of relocation information

You can assign initial values to memory locations 0-476 (which include the interrupt vectors and system communication area) by using an .ASECT assembler directive. They appear in block 0 of the load module, but there are restrictions on the use of ASECTs in this region. You should not perform ASECTs of location 54 or of locations 360-377 because the memory usage map is passed in those locations. In addition, for foreground links, ASECTs of words 52-62 are not permitted because additional parameters are passed to the FRUN command in those locations.

You can set with an .ASECT any location that is not restricted, but be careful if you change the system communication area. The program itself must initialize restricted areas, such as the region 360-377. There are no restrictions on ASECTs if the output format is LDA.

Locations in the region 0-476 might not be loaded at execution time even though your program uses an ASECT to initialize them. For background programs, this is because the R, RUN, and GET commands do not load addresses that are protected by the monitor's memory protection map. For foreground programs, the FRUN command loads only locations 14-22 and 34-50. It ignores all other ASECTs. To initialize a location at run time, use the .PROTECT programmed request. If it is successful, follow it by a MOV instruction.

11.5.3 Load Map

If you request, the linker produces a load map following the completion of the initial pass. This map, shown in Figure 11-1, diagrams the layout of memory for the load module.

The load map lists each program section that is included in the linking process. The line for a section includes the name and low address of the section and its size in bytes. The rest of the line lists the program section attributes, as shown in Table 11-3. The remaining columns contain the global symbols found in the section and their values.

The map begins with the version of the linker, followed by the date and time the program was linked. The second line lists the file name of the program, its title (which is determined by the first module name record in the input file), and the first identification record found. The absolute section is always shown first, followed by any non-relocatable symbols. The modules located in the root segment of the load module list next, followed by those modules that were assigned to overlays in order by their region number (see Section 11.6). Any undefined global symbols then list. The map ends with the transfer address (start address) and high limit or relocatable code in both octal bytes and decimal words.

NOTE

The load map does not reflect the absolute addresses for a REL file that you create to run as a foreground job; you must add the base relocation address determined at FRUN time to obtain the absolute addresses. The linker assumes a base address of 1000.

For example, assume the FRUN command is used to run the program CARL:

```
.FRUN CARL/P
Loaded at 127276
```

The /P option causes FRUN to print the load address, which is 127276 in this example. To calculate the actual location in memory of any global in the program, first subtract 1000 from that global's value. (The value 1000 represents the base address assigned by the linker. This offset is not used at load time.) Then add the result to the load address determined with /P. The final result represents the absolute location of the global. For example, the absolute location of TIME (see Figure 11-1) is 127302 (1004-1000+127276=127302).

```
RT-11 LINK  V03.01       Load Map          Fri 03-Jun-77 18:03:07
CARL  .REL       Title:  DEMOSP  Ident:  V01,03

Section  Addr    Size     Global  Value   Global  Value   Global  Value

. ABS.   000000  001000   (RW,I,GBL,ABS,OVR)
         001000  000036   (RW,I,LCL,REL,CON)
                          TIMBLK  001000  TIME    001004  DBLK    001010
                          LP      001020  AREA    001024  START   001036
                          BUFF    002022

Transfer address = 001036, High limit = 011036 = 2319. words
```

Figure 11-1 Load Map

11.5.4 Library Files

The RT-11 linker can automatically search libraries. Libraries consist of library files, which are specially formatted files produced by the librarian program (described in Chapter 12) that contain one or more object modules. The object modules provide routines and functions to aid you in meeting specific programming needs. (For example, FORTRAN has a set of modules containing all necessary computational functions – SQRT, SIN, COS, etc.). You can use the librarian to create and update libraries. Then you can easily access routines that you use repeatedly or routines that different programs use. Selected modules from the appropriate library file are linked as needed with your program to produce one load module. Libraries are further described in Section 11.7 and in Chapter 12.

NOTE

Library files that you combine with the monitor COPY command or with the PIP /U or /B option are illegal as input to both the linker and the librarian.

11.6 USING OVERLAYS

The ability of RT-11 to use overlays gives you virtually unlimited memory space for an assembly language or FORTRAN program. A program using overlays can be much larger than would normally fit in the available memory space, since portions of the program (called overlay segments) reside on a backup storage device (disk or DECtape).

The RT-11 overlay scheme is a strict multi-region arrangement; it is not tree-structured. Figure 11-3 diagrams this scheme. The overlay system that you construct from your completed program is composed of a root segment that is always memory resident, the current memory-resident overlay segments, and the overlay segments stored on the backup storage device. The root segment is a required part of every overlay program and contains the transfer address, stack space, impure variables, data, and variables needed by many different segments; it must, therefore, never be overlaid. There is a distinct memory area for each overlay region. The overlay segments are brought into memory as they are needed. A segment consists of a set of modules and program sections. Segments that overlay each other must be logically independent; that is, the components of one segment cannot reference the components of another segment with which it shares address space. In addition to being concerned with the logical independence of the overlay segments, you should also consider the general flow of control within the program. Programs execute most efficiently when the system spends only a small amount of execution time (less than 10 percent) overlaying program segments. Figure 11-2 shows a diagram of an overlay scheme. Some examples of overlaid programs are the linker and the FORTRAN compiler.

Overlay segments that share both the same physical memory location and address space form a region. You specify overlay regions to the linker with the /O option as described in Section 11.8.7. The linker calculates the size of any region to be the size of the largest segment within that region. Thus, to reduce the size of a program (that is, the amount of memory it needs), you should first concentrate on reducing the size of the largest segment in each region. The linker creates the overlay regions and edits the program to produce the desired overlays at run-time. Figure 11-3 shows a listing of a diagram of memory showing a link with an overlay structure. Figure 11-4 shows the run-time overlay handler.

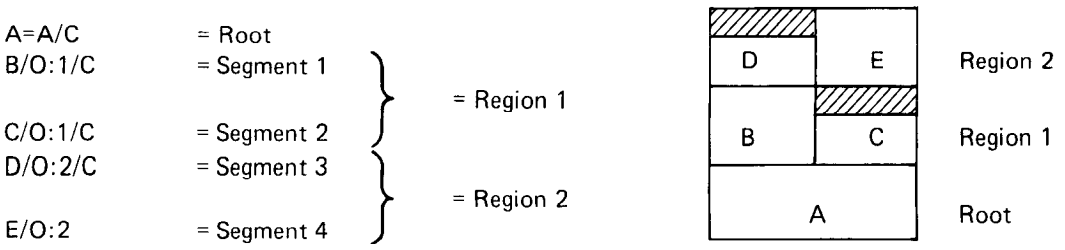


Figure 11-2 Overlay Scheme

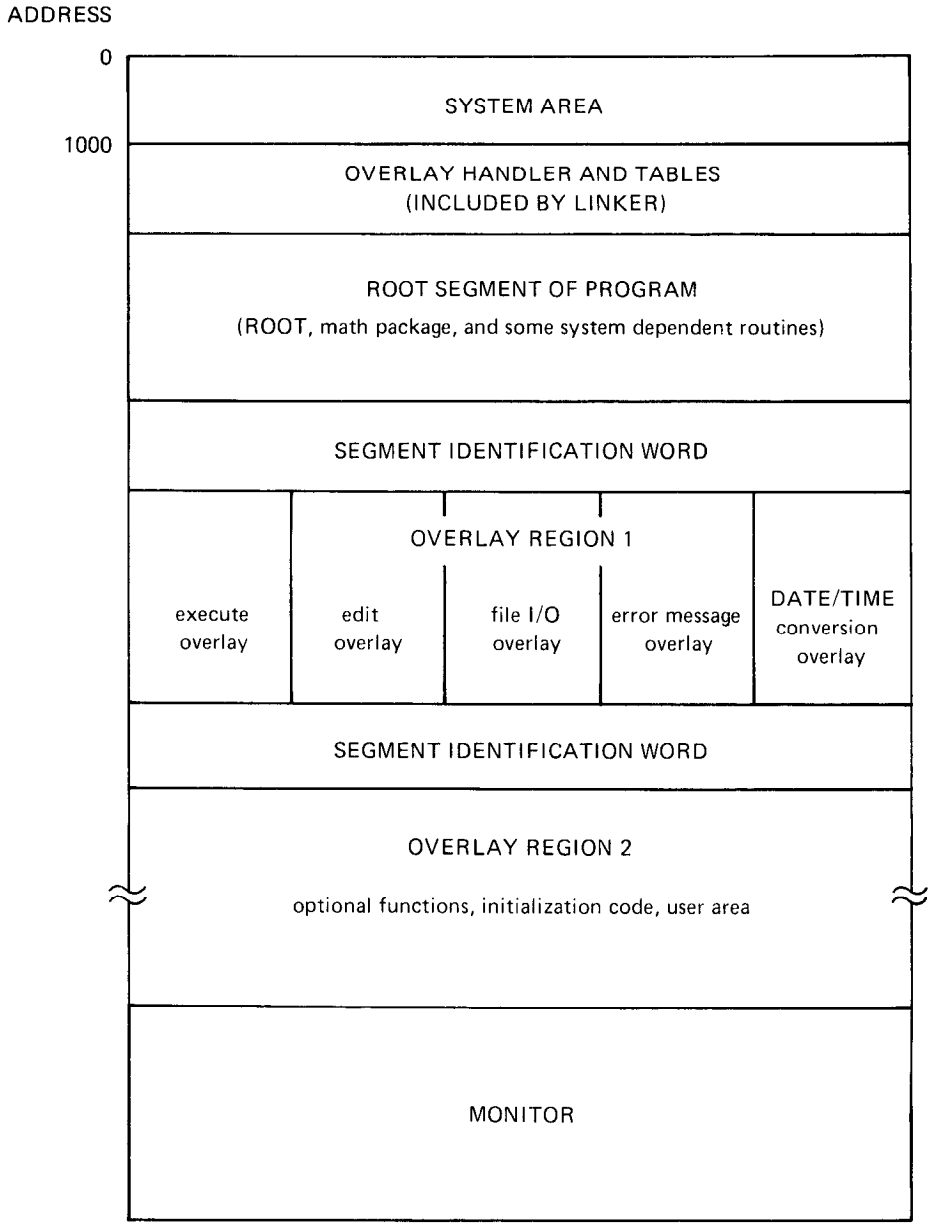


Figure 11-3 Memory Diagram Showing BASIC Link with Overlay Regions

```
        .SBTTL  $OVRH   THE RUN-TIME OVERLAY HANDLER
;THE FOLLOWING CODE IS INCLUDED IN THE USER'S PROGRAM BY THE
;LINKER WHENEVER OVERLAYS ARE REQUESTED BY THE USER.
;THE RUN-TIME OVERLAY HANDLER IS CALLED BY A DUMMY
;SUBROUTINE OF THE FOLLOWING FORM:

;       JSR     R5,$OVRH        ;CALL TO COMMON CODE
;       .WORD   <OVERLAY #>     ;# OF DESIRED SEGMENT
;       .WORD   <ENTRY ADDR>    ;ACTUAL CORE ADDR

;ONE DUMMY ROUTINE OF THE ABOVE FORM IS STORED IN THE RESIDENT PORTION
;OF THE USER'S PROGRAM FOR EACH ENTRY POINT TO AN OVERLAY SEGMENT.
;ALL REFERENCES TO THE ENTRY POINT ARE MODIFIED BY THE LINKER TO INSTEAD
;BE REFERENCES TO THE APPROPRIATE DUMMY ROUTINE.  EACH OVERLAY SEGMENT
;IS CALLED INTO CORE AS A UNIT AND MUST BE CONTIGUOUS IN CORE.  AN
;OVERLAY SEGMENT MAY HAVE ANY NUMBER OF ENTRY POINTS, TO THE LIMITS
;OF CORE MEMORY.  ONLY ONE SEGMENT AT A TIME MAY OCCUPY AN OVERLAY REGION.

.ENABL  LSB
        $OVTAB=1000+$OVRHE-$OVRH
$OVRH:  MOV     R0,-(SP)
        MOV     R1,-(SP)
        MOV     R2,-(SP)
1$:
;       MOV     (R5)+,R0        ;PICK UP OVERLAY NUMBER
        BR      3$              ;FIRST CALL ONLY * * *
        MOV     R0,R1
$OVRHA: ADD     #$OVTAB-6,R1    ;CALC TABLE ADDR
        MOV     (R1)+,R2        ;GET CORE ADDR OF OVERLAY REGION
        CMP     R0,@R2          ;IS OVERLAY ALREADY RESIDENT?
        BEQ     2$              ;YES, BRANCH TO IT
        .READW  17,R2,(R1)+,(R1)+ ;READ FROM OVERLAY FILE
        BCS     5$
2$:     MOV     (SP)+,R2        ;RESTORE USER'S REGS
        MOV     (SP)+,R1
        MOV     (SP)+,R0
        MOV     @R5,R5          ;GET ENTRY ADDRESS
        RTS     R5              ;ENTER OVERLAY ROUTINE AND
                                ;RESTORE USER'S R5

3$:     MOV     #12500,1$       ;RESTORE SWITCH INSTR (MOV (R5)+,R0)
        MOV     (PC)+,R1        ;START ADDR FOR CLEAR OPERATION
$HROOT: .WORD   0               ;HIGH ADDR OF ROOT SEGMENT
        MOV     (PC)+,R2        ;COUNT
$HOVLY: .WORD   0               ;HIGH LIMIT OF OVERLAYS
4$:     CLR     (R1)+           ;CLEAR ALL OVERLAY REGIONS
        CMP     R1,R2
        BLO     4$
        BR      1$              ;AND RETURN TO CALL IN PROGRESS

5$:     EMT     376             ;SYSTEM ERROR 10 (OVERLAY I/O)
        .BYTE   6,373
$OVRHE:
.DSABL  LSB
;OVERLAY SEGMENT TABLE FOLLOWS:
; $OVTAB:       .WORD   <CORE ADDR>,<RELATIVE BLK>,<WORD COUNT>
 [illegible]

;ALSO, THERE IS ONE WORD PREFIXED TO EACH OVERLAY REGION
;THAT IDENTIFIES THE SEGMENT CURRENTLY RESIDENT IN THAT REGION.
```

Figure 11-4 The Run-Time Overlay Handler

You do not need a special code or function call to use overlays. Observe the following rules when you reference parts of your program that might be overlaid.

1. You must make calls or jumps to overlay segments directly to global symbols defined in an instruction p-section (entry points). For example, if ENTER is a global symbol in an overlay segment, the first command is valid, but the second is illegal:

```
JMP ENTER
JMP ENTER+6
```

2. You can use globals defined in an instruction p-section (entry points) of an overlay segment only for transfer of control and not for referencing data within an overlay section. The assembler and linker cannot detect a violation of this rule so they issue no error. However, such a violation can cause the program to use incorrect data. If you reference these global symbols outside of their defining segment, the linker resolves them by using dummy subroutines of four words each in the overlay handler. If such a reference occurs, it is indicated on the load map by a "@" following the symbol.
3. The linker directly resolves global symbols that you define in a data p-section. It is your program's responsibility to load the data into memory before referencing a global symbol defined in a data section. One way to load the data section of an overlay segment is to call an entry point in that segment. This loads the segment if it is not already resident in memory.
4. When you make calls to overlays, the entire return path must be in memory. Observing the following rules accomplishes this:
   a. You can make calls (with expected return) from an overlay segment only to entries in the same segment, the root segment, or an overlay segment with a greater region number.
   b. Calls you make to entries in the same region as the call must be entirely within the same segment, not within another segment in the same region.
   c. You can make jumps (with no expected return) from an overlay segment to any entry in the program. However, jumps should not reference an overlay region whose number is lower than the region from which the last unreturned call was made (for example, if a call was made from region 3, then no jumps should reference regions 1, 2 or 3 until the call has returned).
   d. You can call subroutines in the root segment from overlay segments; in turn, they can call entries from the same overlay segment that called them, or from the root segment, or from another overlay segment with a greater region number. Such subroutines are considered to be part of the overlay segment that called them.
5. You cannot use a p-section (.PSECT or .CSECT) name to pass control to an overlay. It does not load the appropriate segment into memory. (For example, JSR PC,OVSEC is illegal if you use OVSEC as a section name in an overlay.) As stated in 1, above, you must use a global symbol to pass control from one segment to the next. If OVSEC is not a global symbol, the system flags it as an undefined global.
6. Your program cannot use channel 17 (octal) because overlays are read on that channel.
7. You cannot place object modules that are automatically acquired from a library file into overlays. The linker always places those modules in the root segment. However, you can extract modules from a library by using the librarian utility program (see Chapter 12) and then explicitly include them in any segment.
8. You cannot specify library files on the same command line as an overlay. Specify them before you enter any overlay lines.
9. You must specify overlay regions in ascending order. They are read-only. Unlike USR swapping, an overlay handler does not save the segment it is overlaying. Any tables, variables, or instructions that are modified within a given overlay segment are reinitialized to their original values in the SAV or REL file if that segment has been overlaid by another segment. You should place any variables or tables whose values must be maintained across overlays in the root segment.
10. ASECTs at location 1000 or above in an overlay foreground link are illegal; the error message ?LINK-F-Illegal ASECT prints and the link aborts.
11. A global program section that is referenced in more than one segment has its memory allocation in the root segment. This permits common access across the different segments. See Section 11.3.

Note the following information when you write FORTRAN overlays:

1. When you divide a FORTRAN program into a root segment and overlay regions (and subsequently divide each overlay region into overlay segments), you should carefully consider routine placement. Remember that it is illegal to call a routine located in a different overlay segment in the same overlay region, or an overlay region with a lower numeric value (as specified by the linker overlay option, /O:n) from the calling routine. Divide each overlay region into overlay segments that never need to be resident simultaneously. For example, if segments A and B are assigned to region X, they cannot call each other because they occupy the same locations in memory.
2. Place the FORTRAN main program unit in the root segment.
3. In an overlay environment, subroutine calls and function subprogram references can refer only to one of the following:
   a. a FORTRAN library routine (such as ASSIGN, DCOS)
   b. a FORTRAN or assembly language routine contained in the root segment
   c. a FORTRAN or assembly language routine contained in the same overlay segment as the calling routine
   d. a FORTRAN or assembly language routine contained in a segment whose region number is greater than that of the calling routine.
4. In an overlay environment, you must place the COMMON blocks so that they are resident when you reference them. Blank COMMON is always resident because it is always placed in the root segment. You must place all named COMMON either in the root segment or in the segment whose region number is lowest of all the segments that reference the COMMON block. A named COMMON block cannot be referenced by two different segments in the same region unless the COMMON block appears in a segment of a lower region number. The linker automatically places a COMMON block into the root segment if it is referenced by the FORTRAN main program or by a subprogram that is located in the root segment. Otherwise, the linker places a COMMON block in the first segment encountered in the linker command string that references that COMMON block.
5. All COMMON blocks that are initialized with DATA statements must be similarly initialized in the segment in which they are placed.

Refer to the *RT-11/RSTS/E FORTRAN IV User's Guide* for more details.

The ASECT never takes part in overlaying in any way. It is part of the root and is always resident.

The aforementioned sets of rules apply only to communications among the various modules that make up a program. Internally, each module must only observe standard programming rules for the PDP-11 (as described in the *PDP-11 Processor Handbook* and in the *MACRO-11 Language Reference Manual*).

Note that the condition codes set by your program are not preserved on the call across overlay segment boundaries. You can still use the C-bit for error returns.

The linker provides overlay services by including a small resident overlay handler in the same file with your program to be used at program run time. The linker inserts this overlay handler plus some tables into your program beginning at the bottom address. The linker moves your program up in memory by an appropriate amount to make room for the overlay handler and tables, if necessary.

11.7 USING LIBRARIES

You specify libraries in a command string in the same way you specify normal modules; you can include them anywhere in the command string, except in overlay lines. If a global symbol is undefined at the time the linker encounters the library in the input stream, and if a module is included in the library that contains that global definition, then the linker pulls that module from the library and links it into the load image. Only the modules needed to resolve references are pulled from the library; unreferenced modules are not linked.

NOTE

Modules in one library can call modules from another library; however, the libraries must appear in the command string in the order in which they are called. For example, assume module X in library ALIB calls Y from the BLIB library. To correctly resolve all globals, the order of ALIB and BLIB should appear in the command line as:

```
*Z=B,ALIB,BLIB
```

Module B is the root. It calls X from ALIB and brings X into the root. X in turn calls Y which is brought from BLIB into the root.

The linker selectively relocates and links object modules from specific user libraries that were built by the librarian. Figure 11-5 diagrams this general process. During pass-1 the linker processes the input files in the order in which they appear in the input command line. If the linker encounters a library file during pass-1, it makes note of the library in an internal save status block, and then proceeds to the next file. The linker processes only non-library files during the initial phase of pass-1. In the final phase of pass-1 the linker processes only library files. This is when it resolves the undefined globals that were referenced by the non-library files.

The linker processes library files in the order in which they appear in the input command line. The default system library (SYSLIB.OBJ) is always last. The processing steps are as follows:

1. If there are any undefined globals, the linker proceeds to step 2. Otherwise, it skips to step 5.
2. The linker reads as much of the library directory as the input buffer can hold.
3. The linker then searches the entire list of undefined globals for a match with the library directory. It places any globals that match in an internal library module list. If more of the library directory remains to be read, the linker proceeds to step 2.
4. The linker now processes the modules from the library that are associated with the matching undefined globals. If this processing results in new undefined globals that can be resolved by the current library, the linker goes back to step 2.
5. The linker closes the current library and processes the next library file, starting with step 1.

This search method allows modules to appear in any order in the library. You can specify any number of libraries in a link and they can be positioned anywhere, with the exception of forward references between libraries. The default system library, SYSLIB.OBJ, is the last library file the linker searches to resolve any remaining undefined globals.

Some languages, such as FORTRAN, have an Object Time System (OTS) that the linker takes from a library and includes in the final module. The most efficient way to accomplish this is to include these OTS routines (such as NHD, OTSCOM, and V2NS for FORTRAN) in SYSLIB.OBJ.

Libraries are input to the linker in the same way as other input files. Here is a sample LINK command string:

```
*TASK01,LP:=MAIN,MEASUR
```

This causes program MAIN.OBJ to be read from DK: as the first input file. Any undefined symbols generated by program MAIN.OBJ should be satisfied by the library file MEASUR.OBJ specified in the second input file. The linker tries to satisfy any remaining undefined globals from the default library, SYSLIB.OBJ. The load module, TASK01.SAV, is stored on DK: and a load map prints on the line printer.

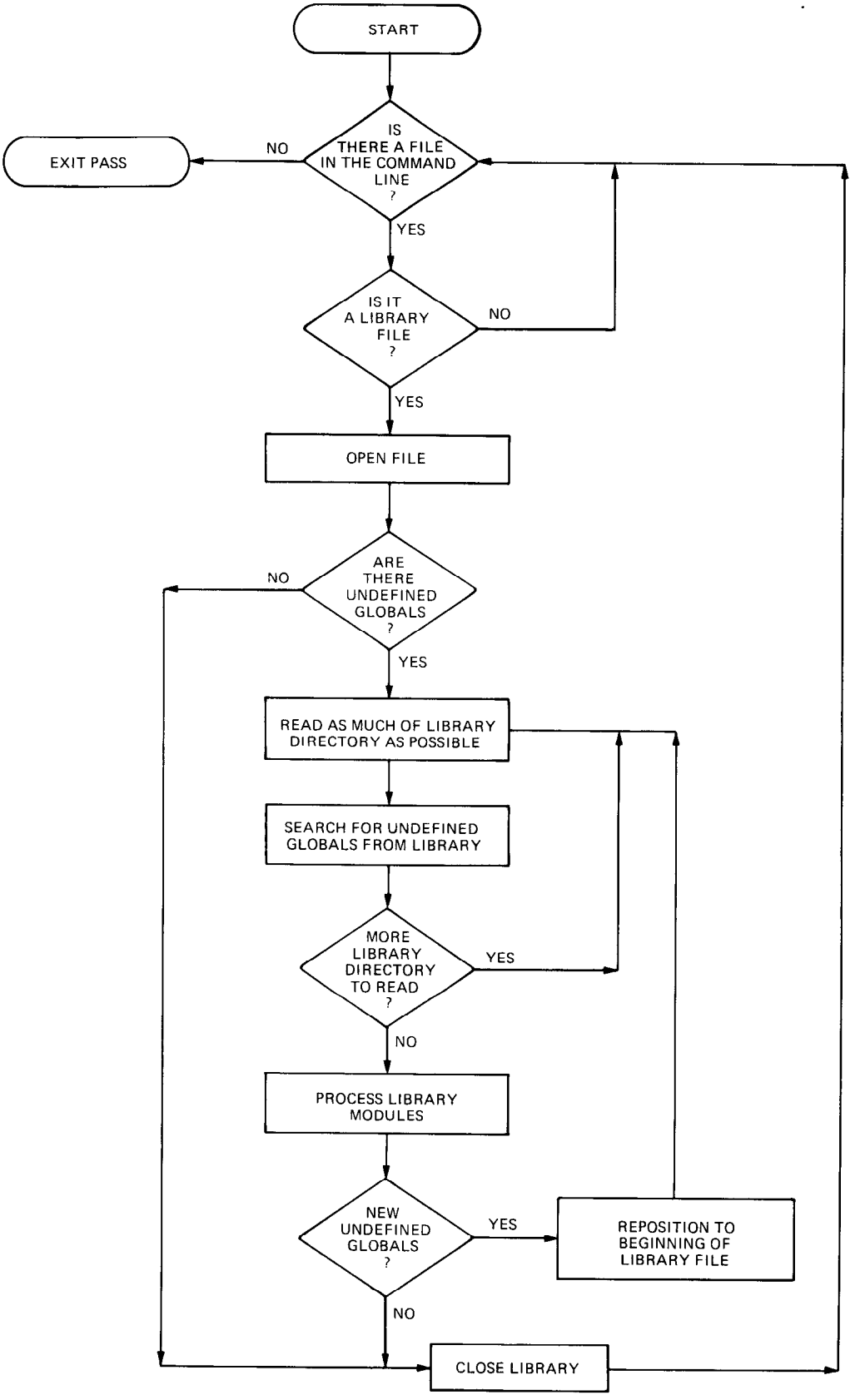


Figure 11-5 Library Searches

11.8 OPTION DESCRIPTIONS

The options summarized in Table 11-2 are described in detail below.

11.8.1 Bottom Address Option (/B:n)

The /B:n option supplies the lowest address to be used by the relocatable code in the load module. The argument, n, is a 6-digit unsigned octal number that defines the bottom address of the program being linked. If you do not supply a value for n, the linker prints:

```
?LINK-F-/B No value
```

Retype the command, supplying an even octal value.

When you do not specify /B, the linker positions the load module so that the lowest address is location 1000 (octal). If the ASECT size is greater than 1000, the size of ASECT is used.

If you supply more than one /B option during the creation of a load module, the linker uses the first /B option specification. /B is illegal when you are linking to a high address (/H). /B is also illegal with foreground links. These modules are always linked to a bottom address of 1000 (octal).

NOTE

The bottom value must be an unsigned even octal number. If the value is odd, the ?LINK-F-/B odd value error message prints. Reenter the command string specifying an unsigned even octal number as the argument to the /B option.

The following command causes the relocatable code from the input file to be linked starting at location 500 (octal).

```
*OUTPUT,LP:=INPUT/B:500
```

11.8.2 Continue Option (/C) or (//)

The continue option (/C) lets you type additional lines of command string input. Use the /C option at the end of the current line and repeat it on subsequent command lines as often as necessary to specify all the input modules in your program. Do not enter a /C option on the last line of input.

The following command indicates that input is to be continued on the next line; the linker prints an asterisk.

```
*OUTPUT,LP:=INPUT/C
*
```

An alternate way to enter additional lines of input is to use the // option on the first line. The linker continues to accept lines of input until it encounters another // option, which can be either on a line with input file specifications, or on a line by itself. The advantage of using the // option instead of the /C option is that you do not have to type the // option on each continuation line. This example shows how the linker itself is linked:

```
*LINK,LINK=LINKO/B:700/W//
*LNKOV1/O:1
*LNKGSD/O:1
*LNKHDR/O:1
*LNKMAP/O:1
*LNKSAV/O:1
*LNKEM/O:1
*//
```

You cannot use the /C option and the // option together in a link command sequence. That is, if you use // on the first line, you must use // to terminate input on the last line. If you use /C on the first line, use /C on all lines but the last.

11.8.3 Extend Program Section Option (/E:n)

The /E:n option allows you to extend a program section to a specific value. Type the /E:n option at the end of the first command line. After you have typed all input command lines, the linker prompts with:

```
Extend section?
```

Respond with the name of the program section to be extended. The resultant program section size is equal to or greater than the value you specify depending upon the space the object code actually requires. Note that you can extend only one section.

The following example extends section CODE to 100 (octal) blocks.

```
*X,TT:=LK001/E:100
Extend section? CODE
```

11.8.4 Default FORTRAN Library Option (/F)

By indicating the /F option in the command line, you can link the FORTRAN library (FORLIB.OBJ on the system device SY:) with the other object modules you specify. You do not need to specify FORLIB explicitly. For example:

```
*FILE,LP:=AB/F
```

The object module AB.OBJ from DK: and the FORTRAN library SY:FORLIB.OBJ are linked together to form a load module called FILE.SAV.

The linker automatically searches a default system library, SY:SYSLIB.OBJ. The library normally includes the modules that compose FORLIB. The /F option is provided only for compatibility with other versions of RT-11. You should not have to use /F.

11.8.5 Highest Address Option (/H:n)

The /H:n option allows you to specify the top (highest) address to be used by the relocatable code in the load module. The argument, n, represents an unsigned even octal number. If you do not specify n, the linker prints:

```
?LINK-F-/H no value
```

Retype the command, supplying an even octal number to be used as the value.

If you specify an odd value, the linker responds with:

```
?LINK-F-/H odd value
```

Retype the command, supplying an even octal number.

If the value is not large enough to accommodate the relocatable code, the linker prints:

```
?LINK-F-/H value too low
```

Relink the program with a larger value.

The /H option cannot be used with the /R or /Y or /B options.

NOTE

Be careful when you use the /H option. Most RT-11 programs use the free memory above the relocatable code as a dynamic working area for I/O buffers, device handlers, symbol tables, etc. The size of this area differs on different memory configurations. Programs linked to a specific high address might not run in a system with less physical memory because there is less free memory.

11.8.6 Include Option (/I)

The /I option lets you take global symbols from any library and include them in the linking process even when they are not needed to resolve globals. This provides a method for forcing modules that are not called by other modules to be loaded from the library. When you specify the /I option, the linker prints:

```
Library search?
```

Reply with the list of global symbols to be included in the load module: type a carriage return to enter each symbol in the list. A carriage return alone terminates the list of symbols.

The following example includes the global $SHORT in the load module:

```
*SCCA=RK1:SCCA/I
Library search?  $SHORT
Library search?
```

11.8.7 Memory Size Option (/K:n)

The /K:n option lets you insert a value into word 56 of block 0 of the image file. The argument, n, represents the number of 1K blocks of memory required by the program; n is an integer in the range 1-28. You cannot use the /K option with the /R option. The /K:n option is provided mainly for compatibility with the RSTS operating system. You should not need to use it with RT-11.

11.8.8 LDA Format Option (/L)

The /L option produces an output file in LDA format instead of memory image format. The LDA format file can be output to any device including those that are not block-replaceable, such as paper tape or cassette. It is useful for files that are to be loaded with the Absolute Loader. The default file type .LDA is assigned when you use the /L option. You cannot use the /L option with the overlay option (/O) or the foreground link option (/R). The following example links files IN and IN2 on device DK: and outputs an LDA format file OUT.LDA to the cassette and a load map to the line printer.

```
*CT:OUT,LP:=IN,IN2/L
```

11.8.9 Modify Stack Address Option (/M[:n])

The stack address, location 42, is the address that contains the initial value for the stack pointer. The /M option lets you specify the stack address. The argument, n, is an even, unsigned 6-digit octal number that defines the stack address. After all input lines have been typed, the linker prints the following message if you have not specified a value for n:

```
Stack symbol?
```

In this case, specify the global symbol whose value is the stack address. You cannot specify a number. If you specify a nonexistent symbol, an error message prints and the stack address is set to the system default (1000 for SAV files) or to the bottom address if you used /B. If the program's absolute section extends beyond location 1000, the default stack space starts after the largest .ASECT contribution.

Direct assignment (with .ASECT) of the stack address within the program takes precedence over assignment with the /M option. The statements to do this in a MACRO program are as follows:

```
.ASECT
.=42
.WORD   INITSP   ;INITIAL STACK SYMBOL VALUE
.PSECT           ;RETURN TO PREVIOUS SECTION
```

The following example modifies the stack address.

```
*OUTPUT=INPUT/M
Stack symbol? BEG
```

11.8.10 Overlay Option (/O:n)

The /O option segments the load module so that the entire program is not memory resident at one time. This lets you execute programs that are larger than the available memory. The argument, n, is an unsigned octal number (up to six digits in length) specifying the overlay region to which the module is assigned. The /O option must follow (on the same line) the specification of the object modules to which it applies, and only one overlay region can be specified on a command line. Overlay regions cannot be specified on the first command line; that is reserved for the root segment. You must use /C or // for continuation.

You specify co-resident overlay routines (a group of subroutines that occupy the overlay region and segment at the same time) as follows:

```
*OBJA,OBJB,OBJC/O:1/C
*OBJD,OBJE/O:2/C
  .
  .
  .
```

All modules that the linker encounters until the next /O option will be co-resident overlay routines. If you specify, at a later time, the /O option with the same value you used previously (same overlay region), then the linker opens up the corresponding overlay area for a new group of subroutines. The new group of subroutines occupy the same locations in memory as the first group, but not at the same time. For example, if subroutines in object modules R and S are to be in memory together, but are never needed at the same time as T, then the following commands to the linker make R and S occupy the same memory as T (but at different times):

```
*MAIN,LP:=ROOT/C
*R,S/O:1/C
*T/O:1
```

The example shown above can also be written as follows:

```
*MAIN,LP:=ROOT/C
*R/O:1/C
*S/C
*T/O:1
```

The following example establishes two overlay regions.

```
*OUTPUT,LP:=INPUT//
*OBJA/O:1
*OBJB/O:1
*OBJC/O:2
*OBJD/O:2
*//
```

You must specify overlays in ascending order by region number. For example:

```
*A=A/C
*B/O:1/C
*C/O:1/C
*D/O:1/C
*E,F/O:2/C
*G/O:2
```

The following overlay specification is illegal since the overlay regions are not given in ascending numerical order (an error message prints in each case):

```
*X=LIBRO//
*LIBR1/O:1
*LIBR2/O:O
?LINK-W-/O ignored
*//
```

In the above example, the overlay option immediately preceding the error message is ignored.

11.8.11 Library List Size Option (/P:n)

The /P:n option lets you change the amount of space allocated for the library routine list. Normally, the default value allows enough space for your needs. It reserves space for approximately 256 unique library routines, which is the equivalent of specifying /P:256. (decimal) or /P:400 (octal).

The error message ?LINK-F-Library list overflow, increase size with /P indicates that you need to allocate more space for the library routine list. You must relink the program that makes use of the library routines. Use the /P:n option and supply a value for n that is greater than 256.

You can use the /P:n option to correct for symbol table overflow. Specify a value for n that is less than 256. This reduces the space used for the library routine list and increases the space allocated for the symbol table. If the value you choose is too small, the ?LINK-F-Library list overflow, increase size with /P message prints. In the following command, the amount of space for the library routine list is increased to 300 (decimal).

```
*SCCA=RK1:SCCA/P:300.
```

11.8.12 REL Format Option (/R[:n])

The /R[:n] option produces an output file in REL format for use as a foregound job with the FB or XM monitor. You cannot use .REL files with the SJ monitor. The /R option assigns the default file type .REL to the output file. The optional argument, n, represents the amount of stack space to allocate for the foreground job. The default value is 128. (decimal) bytes of stack space. If you also use the /M option, the value or global symbol associated with it overrides the /R value.

The following command links files FILEI.OBJ and NEXT.OBJ and stores the output on DT2: as FILEO.REL. It also prints a load map on the line printer.

```
*DT2:FILEO,LP:=FILEI,NEXT/R:200
```

You cannot use the /B, /H, and /L options with /R since a foreground REL job has a temporary bottom address of 1000 and is always relocated by FRUN. An error message prints if you attempt this. The /K option is also illegal with /R.

11.8.13 Symbol Table Option (/S)

The /S option instructs the linker to allow the largest possible memory area for its symbol table at the expense of input and output buffer space, which makes the linking process slower. You should use the /S option only if an attempt to link a program failed because of symbol table overflow. Often, use of /S allows the program to link.

11.8.14 Transfer Address Option (/T[:n])

The transfer address is the address at which a program starts when you initiate execution with an R, RUN, or FRUN command. It prints on the last line of the load map. The /T option lets you specify the start address of the load module. The argument, n, is a six-digit unsigned octal number that defines the transfer address. If you do not specify n, the following message prints:

```
Transfer symbol?
```

In this case, specify the global symbol whose value is the transfer address of the load module. Terminate your response with a carriage return. You cannot specify a number in answer to this message. If you specify a nonexistent symbol, an error message prints and the transfer address is set to 1 so that the program traps immediately if you attempt to execute it. If the transfer address you specify is odd, the program does not start after loading and control returns to the monitor.

Direct assignment (.ASECT) of the transfer address within the program takes precedence over assignment with the /T option. The transfer address assigned with a /T has precedence over that assigned with an .END assembly directive. To assign the transfer address within a MACRO program, use statements similar to these:

```
            .ASECT
            .=40
            .WORD     START1   ;SYMBOL VALUE FOR TRANSFER ADDRESS
            .PSECT             ;RETURN TO PREVIOUS SECTION

START1:     .
            .
            .
            or
START2:     .                  ;SECONDARY STARTING ADDRESS
            .
            .
            .END      START2
```

The following example links the files LIBR0.OBJ and ODT.OBJ together and starts execution at ODT's transfer address.

```
*LBRODT,LBRODT=LIBR0,ODT/T/W//
*LIBR1/O:1
*LIBR2/O:1
*LIBR3/O:1
*LIBR4/O:1
*LIBR5/O:1
*LBREM/O:1//
Transfer symbol? O.ODT
*
```

11.8.15 Round Up Option (/U:n)

The /U:n option rounds up the section you specify so that the size of the root segment is a whole number multiple of the value you supply. The argument, n, must be a power of 2. When you specify the /U:n option, the linker prompts:

```
Round section?
```

Reply with the name of the program section to be rounded. The program section must be in the root segment. Note that you can round only one program section. The following example rounds up section CHAR.

```
*LK007,TT:=LK007/U:200
Round section?  CHAR
```

If the program section you specify cannot be found, the linker prints ?LINK-W-Round section not found. The linking process continues with no rounding.

11.8.16 Map Width Option (/W)

The /W option directs the linker to produce a wide load map listing. If you do not specify the /W option, the listing is wide enough for three GLOBAL VALUE columns (normal for paper with 80 columns). If you use the /W command, the listing is six columns wide, which is ideal for a 132 column page.

11.8.17 Bit Map Inhibit Option (/X)

The /X option instructs the linker not to output the bit map if code is below 400. This option is provided only for compatibility with the RSTS operating system. The bit map is stored in locations 360-377 in block 0 of the load module. The linker normally stores the program memory usage bits in these eight words. Each bit represents one 256-word block of memory. This information is used by the R, RUN and GET commands when loading the program; therefore, use care when you use this option.

11.8.18 Boundary Option (/Y:n)

The /Y:n option starts a specific program section on a particular address boundary. The linker generates a whole number multiple of n, the value you specify, for the starting address of the program section. The argument, n, must be a power of 2. The linker extends the size of the previous program section to accommodate the new starting address. When you have entered all the input lines, the linker prompts:

```
Boundary section?
```

Respond with the name of the program section whose starting address you are modifying. Terminate your response with a carriage return. Note that you can specify only one program section for this option. If the program section you specify cannot be found, the linker prints ?LINK-W-Boundary section not found. The linking process continues.

The RT-11 monitors have internal 2-block overlays. The first overlay segment, OVLY0, must start on a disk block boundary:

```
*RKMNSJ.SYS=RKBTSJ,RT11SJ,RKTBSJ,RK/Y:1000
Boundary Section?  OVLY0
```

11.8.19 Zero Option (/Z:n)

The /Z:n option fills unused locations in the load module and places a specific value in these locations. The argument, n, represents the value to be placed in the unused locations. This option can be useful in eliminating random results that occur when the program references uninitialized memory by mistake. The system automatically zeroes unused locations. Use the /Z:n option only when you want to store a value other than zero in unused locations.

11.9 LINKER PROMPTS

Some of the linker operations prompt for more information, such as the names of specific global symbols or sections. The linker issues the prompt after you have entered all the input specifications, but before the actual linking begins. Table 11-6 shows the sequence in which the prompts occur.

Table 11-6 Linker Prompting Sequence

Prompt	Option
Transfer symbol?	/T
Stack symbol?	/M
Extend section?	/E:n
Boundary section?	/Y:n
Round section?	/U:n
Library search?	/I

The library search prompt is last because it can accept more than one symbol and is terminated by a carriage return on a line by itself.

Note that if the command lines are in an indirect file and the linker encounters an end-of-file before the prompting information has been supplied, it prints the prompt messages on the terminal.

The following example shows how the linker prompts for information when you combine options.

```
*LK001=LK001/T/M/E:100/Y:400/U:20/I
Transfer symbol? 0.ODT
Stack symbol? ST3
Extend section? CHAR
Boundary section? CODE
Round section? STKSP
Library search? $SHORT
Library search?
*
```

CHAPTER 12
LIBRARIAN (LIBR)

The librarian utility program (LIBR) lets you create, update, modify, list, and maintain object library files. It also lets you create macro library files to use with the V03 MACRO-11 assembler.

A library file is a direct access file (a file that has a directory) that contains one or more modules of the same module type. The librarian organizes the library files so that the linker and MACRO-11 assembler can access them rapidly. Each library contains a library header, library directory (or global symbol or macro name table), and one or more object modules or macro definitions. The object modules in a library file can be routines that are repeatedly used in a program, routines that are used by more than one program, or routines that are related and simply gathered together for convenience. The contents of the library file are determined by your needs. An example of a typical object library file is the default system library that the linker uses, SYSLIB.OBJ. An example of a macro library file is SYSMAC.SML, which MACRO uses automatically to process programmed requests.

You access object modules in a library file from another program by making calls or references to their global symbols; you then link the object modules with the program that uses them, producing a single load module (see Chapter 11).

Consult the *RT-11 Software Support Manual* for more information on the internal data structure of a library file. However, that information is not necessary for your understanding of this chapter.

12.1 CALLING AND USING LIBR

To call the RT-11 librarian from the system device, respond to the dot (.) printed by the keyboard monitor by typing:

```
R LIBR (RET)
```

The Command String Interpreter prints an asterisk at the left margin on the console terminal when it is ready to accept a command line. Chapter 6, Command String Interpreter, describes the general syntax of the command line LIBR accepts.

Type two CTRL/Cs to halt the librarian at any time (or a single CTRL/C to halt the librarian when it is waiting for console terminal input) and return control to the monitor. To restart the librarian, type R LIBR or REENTER in response to the monitor's dot.

Section 12.2 explains how to use the librarian to create and maintain object libraries; Section 12.3 describes how to create macro libraries.

Specify the LIBR command string in the following general format:

```
library-filespec[n],list-filespec[n]=input-filespecs/options
```

where

library-filespec[n]	represents the library file to be created or updated. The optional argument, n, represents the number of blocks to allocate for the output file.

list-filespec[n]	represents a listing file for the library's contents. The optional argument, n, represents the number of blocks to allocate for the listing file.
input-filespec	represents the input object modules (you can specify up to six input files); it can also represent a library file to be updated.
option	represents an option from Table 12-1.

You specify devices and file names in the standard RT-11 command string syntax, with default file types assigned as follows:

File	File Type
list file:	.LST
library file:	.OBJ
input files:	.OBJ

If you do not specify a device, the default device (DK:) is assumed.

Each input file consists of one or more object modules and is stored on a given device under a specific file name and file type. Once you insert an object module into a library file you no longer reference the module by the name of the file of which it was a part; instead, you reference it by its individual module name. You assign this module name with the assembler with either a .TITLE statement in the assembly source program, or with the default name .MAIN. upon absence of a .TITLE statement or the subprogram name for FORTRAN routines. Thus, for example, the input file FORT.OBJ can exist on DT2: and can contain an object module called ABC. Once you insert the module into a library file, reference only ABC (not FORT.OBJ).

The input files normally do not contain main programs but rather subprograms, functions, and subroutines. The library file must never contain a FORTRAN "BLOCK DATA" subprogram; there is no undefined global symbol to cause the linker to load it automatically.

12.2 OPTION COMMANDS AND FUNCTIONS FOR OBJECT LIBRARIES

You maintain object library files by using option commands. Functions that you can perform include object module deletion, insertion and replacement, library file creation, and listing of an object library file's contents.

Table 12-1 summarizes the options available for you to use with RT-11 LIBR for object libraries. The following sections, which are arranged alphabetically by option, describe the options in greater detail.

Table 12-1 LIBR Object Options

Option	Command Line	Section	Meaning
/C	any but last	12.2.1	Command continuation; allows you to type the input specification on more than one line.
/D	first	12.2.4	Delete; deletes modules that you specify from a library file.
/E	first	12.2.5	Extract; extracts a module from a library and stores it in an .OBJ file.
/G	first	12.2.6	Global deletion; deletes global symbols that you specify from the library directory.
/N	first	12.2.7	Names; includes the module names in the directory.

(Continued on next page)

Table 12-1 (Cont.) LIBR Object Options

Option	Command Line	Section	Meaning
/P	first	12.2.8	P-section names; includes the program section names in the directory.
/R	first	12.2.9	Replace; replaces modules in a library file. This option must follow the file specification to which it applies.
/U	first	12.2.10	Update; inserts and replaces modules in a library file. This option must follow the file specification to which it applies.
/W	first	12.2.11	Indicates wide format for the listing file.
//	first and last	12.2.1	Command continuation; allows you to type the input specification on more than one line.

There is no option to indicate module insertion. If you do not specify an option, the librarian automatically inserts modules into the library file.

12.2.1 Command Continuation Options (/C and //)

You must use a continuation option whenever there is not enough room to enter a command string on one line. The maximum number of input files that you can enter on one line is six; you can use the /C option or the // option to enter more. Type the /C option at the end of the current line and repeat it at the end of subsequent command lines as often as necessary, so long as memory is available; if you exceed memory, an error message prints. Each continuation line after the first command line can contain only input file specifications (and no other options). Do not specify a /C option on the last line of input. If you use the // option, type it at the end of the first input line and again at the end of the last input line.

The following example creates a library file on the default device (DK:) under the file name ALIB.OBJ; it also creates a listing of the library file's contents as LIBLST.LST (also on the default device). The file names of the input modules are MAIN.OBJ, TEST.OBJ, FXN.OBJ, and TRACK.OBJ, all from DT1:.

```
*ALIB,LIBLST=DT1:MAIN,TEST,FXN/C
*DT1:TRACK
```

The next example creates a library file on the default device (DK:) under the name BLIB.OBJ. It does not produce a listing. Input files are MAIN.OBJ from the default device, TEST.OBJ from RK1:, FXN.OBJ from RK0:, and TRACK.OBJ from DT1:.

```
*BLIB=MAIN//
*RK1:TEST
*RK0:FXN
*DT1:TRACK//
```

Another way of writing this command line is:

```
*BLIB=MAIN,RK1:TEST,RK0:FXN//
*DT1:TRACK
*//
```

12.2.2 Creating a Library File

To create a library file, specify a file name on the output side of a command line.

The following example creates a new library called NEWLIB.OBJ on the default device (DK:). The modules that make up this library file are in the files FIRST.OBJ and SECOND.OBJ, both on the default device.

```
*NEWLIB=FIRST,SECOND
```

12.2.3 Inserting Modules into a Library

Whenever you specify an input file without specifying an associated option, the librarian inserts the modules in the file into the library file you name on the output side of the command string. You can specify any number of input files. If you include section names (if you use /P) in the global symbol table and if you attempt to insert a file that contains a global symbol or PSECT (or CSECT) having the same name as a global symbol or PSECT already existing in the library file, the librarian prints a warning message. The librarian does, however, update the library file, ignore the global symbol or section name in error, and return control to the CSI. You can then enter another command string.

Although you can insert object modules that exist under the same name (as assigned by the .TITLE statement), this practice is not recommended because of the difficulty and ambiguity involved when you need to update these modules (Sections 12.2.2.9 and 12.2.2.10 describe replacing and updating).

NOTE

The librarian performs module insertion, replacement, deletion, merge, and update concurrently with creating the library file. Therefore, you must indicate the library file to which the operation is directed on both the input and output sides of the command line, since effectively the librarian creates a "new" output library file each time it performs one of those operations. You must specify the library file first in the input field.

The following command line inserts the modules included in the files FA.OBJ, FB.OBJ, and FC.OBJ on DT1: into a library file named DXYNEW.OBJ on the default device. The resulting library also includes the contents of library DXY.OBJ.

```
*DXYNEW=DXY,DT1:FA,FB,FC
```

The next command line inserts the modules contained in files THIRD.OBJ and FOURTH.OBJ into the library NEWLIB.OBJ.

```
*NEWLIB,LIST=NEWLIB,THIRD,FOURTH
```

Note that the resulting library contains the original library plus some new modules. Note also that the resulting library replaces the original library because the same name was used in this example for the input and output library.

12.2.4 Delete Option (/D)

The /D option deletes modules and all their associated global symbols from the library.

When you use the /D option, the librarian prompts:

```
Module name?
```

Respond with the name of the module to be deleted followed by a carriage return; continue until you have entered all modules to be deleted. Type a carriage return immediately after the Module name? message to terminate input and initiate execution of the command line.

The following example deletes the modules SGN and TAN from the library file TRAP.OBJ on DT3:.

```
*DT3:TRAP=DT3:TRAP/D
Module name? SGN
Module name? TAN
Module name?
```

The next example deletes the module FIRST from the library LIBFIL.OBJ; all modules in the file ABC.OBJ replace old modules of the same name in the library; it also inserts the modules in the file DEF.OBJ into the library.

```
*LIBFIL=LIBFIL/D,ABC/R,DEF
Module name? FIRST
Module name?
```

In the following example, the librarian deletes two modules of the same name from the library file LIBFIL.OBJ.

```
*LIBFIL=LIBFIL/D
Module name? X
Module name? X
Module name?
```

12.2.5 Extract Option (/E)

The /E option allows you to extract an object module from a library file and place it in an .OBJ file.

When you specify the /E option, the librarian prints:

```
Global?
```

Respond with the name of the object module to be extracted. If you specify a global name, the librarian extracts the entire module of which that global is a part.

You cannot use the /E option on the same command line with any other option.

The following example extracts the ATAN routine from the FORTRAN library, SYSLIB.OBJ, and stores it in a file called ATAN.OBJ on DX1:.

```
*DX1:ATAN=SYSLIB/E
Global? ATAN
Global?
```

The next example extracts the $PRINT routine from SYSLIB.OBJ and stores it on DM1: as PRINT.OBJ.

```
*DM1:PRINT=SYSLIB/E
Global? $PRINT
Global?
```

The extract option is particularly useful if you need to use a routine in only one overlay segment. Normally, all modules that the linker acquires automatically from a library go into the root segment. To circumvent this, you can extract a routine with /E, then link it into an overlay segment instead of into the root segment.

12.2.6 Delete Global Option (/G)

The /G option lets you delete a specific global symbol from a library file's directory.

When you use the /G option, the librarian prints:

```
Global?
```

Respond with the name of the global symbol to be deleted followed by a carriage return; continue until you have entered all globals to be deleted. Type a carriage return immediately after the Global? message to terminate input and initiate execution of the command line.

The following command instructs the librarian to delete the global symbols NAMEA and NAMEB from the directory found in the library file ROLL.OBJ on DK:.

```
*ROLL=ROLL/G
Global? NAMEA
Global? NAMEB
Global?
```

The librarian deletes globals only from the directory (and not from the library itself). Whenever you update a library file all globals that you previously deleted are restored unless you use the /G option again to delete them. This feature lets you recover if you inadvertently delete the wrong global.

12.2.7 Include Module Names Option (/N)

The librarian does not include module names in the directory unless you use the /N option on the first line of the command. The linker loads modules from libraries based on undefined globals, not on module names. The linker also provides equivalent functions by using global symbols and not module names. Normally, then, it is a waste of space and a performance compromise to include module names in the directory.

If you do not include module names in the directory, the MODULE column of the directory listing is blank unless the module requires a continuation line to print all its globals. A plus (+) sign in the MODULE column indicates continued lines. The /N option is useful mainly when you create a temporary library in order to obtain a directory listing.

If the library does not have module names in its directory, you must create a new library to include the module names. The following example illustrates how to do this. It creates a temporary new library from the current library (by specifying the null device for output) and lists its directory on the terminal. The current library OLDLIB remains unchanged.

```
*NL:TEMP,TT:=OLDLIB/N
RT-11 LIBRARIAN V03.00   TUE 03-MAY-77 20:36:41
TEMP                     TUE 03-MAY-77 20:36:40

MODULE          GLOBALS          GLOBALS          GLOBALS

IRAD50          IRAD50           RAD50
JMUL            JMUL
LEN             LEN
SUBSTR          SUBSTR
JADD            JADD
JCMP            JCMP
```

12.2.8 Include P-section Names Option (/P)

The librarian does not include program section names in the directory unless you use the /P option on the first line of the command. The linker does not use section names to load routines from libraries; including the names can decrease linker performance. Including program section names also causes a conflict in the library directory and subsequent searches, since the librarian treats section names and global symbols identically.

This option is provided for compatibility with RT-11V2C. DIGITAL recommends that you avoid using it with RT-11V03.

12.2.9 Replace Option (/R)

Use the /R option to replace modules in a library file. The /R option replaces existing modules in the library file you specify as output with the modules of the same names contained in the file(s) you specify as input. In the command string, enter the input library file before the files used in the replacement operation.

If an old module does not exist under the same name as an input module, or if you specify the /R option on a library file, the librarian prints an error message preceded by the module name, and ignores the replace command. /R must follow each input file name containing modules for replacement.

The following command line indicates that the modules in the file INB.OBJ are to replace existing modules of the same names in the library file TFIL.OBJ. The object modules in the files INA.OBJ and INC.OBJ are to be added to TFIL. All files are to be stored on the default device DK:.

```
*TFIL=TFIL,INA,INB/R,INC
```

The same operation occurs in the next command as in the preceding example, except that this updated library file is assigned the new name XFIL.

```
*XFIL=TFIL,INA,INB/R,INC
```

12.2.10 Update Option (/U)

The /U option lets you update a library file by combining the insert and replace functions. If the object modules that compose an input file in the command line already exist in the library file, the librarian replaces the old modules in the library file with the new modules in the input file. If the object modules do not already exist in the library file, the librarian inserts those modules into the library. (Note that some of the error messages that might occur with separate insert and replace functions do not print when you use the update function.) /U must follow each input file that contains modules to be updated. Specify the input library file before the input files in the command line.

The following command line instructs the librarian to update the library file BALIB.OBJ on the default device. First the modules in FOLT.OBJ and BART.OBJ replace old modules of the same names in the library file, or if none already exist under the same names, the modules are inserted. The modules from the file TAL.OBJ are then inserted; an error message prints if the name of the module in TAL.OBJ already exists.

```
*BALIB=BALIB,FOLT/U,TAL,BART/U
```

In the next example, there are two object modules of the same name (X) in both Z and XLIB; these are first deleted from XLIB. This ensures that both the modules called X in file Z are correctly placed into the library. Globals SEC1 and SEC2 are also deleted from the directory but automatically return the next time the library XLIB.OBJ is updated.

```
*XLIB=XLIB/D,Z/U/G
Module name?  X
Module name?  X
Module name?
Global?  SEC1
Global?  SEC2
Global?
```

12.2.11 Wide Option (/W)

The /W option gives you a wider listing if you request a listing file. The wider listing has six GLOBAL columns instead of three, as in the normal listing. This is useful if you list the directory on a line printer or a terminal that has 132 columns.

12.2.12 Listing the Directory of a Library File

You can request a listing of the contents of a library file (the global symbol table) by indicating both the library file and a list file in the command line. Since a library file is not being created or updated, you do not need to indicate the file name on the output side of the command line; however, you must use a comma to designate a null output library file.

The command syntax is as follows:

```
*,LP:=library-filespec
```

or

```
*,list-filespec=library-filespec
```

where

library-filespec	represents the existing library file.
LP:	indicates that the listing is to be sent directly to the line printer (or terminal, if you use TT:).
list-filespec	represents a list file of the library file's contents.

The following command outputs to DT2: as LIST.LST, a listing of the contents of the library file LIBFIL.OBJ on the default device.

```
*,DT2:LIST=LIBFIL
```

The next command sends to the line printer a listing of all modules in the library file FLIB.OBJ, which is stored on the default device.

```
*,LP:=FLIB
```

Here is a sample section of a large directory listing:

```
*,TT:=SYSLIB
RT-11 LIBRARIAN V03.00   TUE 03-MAY-77 21:01:01
SYSLIB                   TUE 03-MAY-77 20:59:47

MODULE          GLOBALS         GLOBALS         GLOBALS

                DCO$            ECO$            FCO$
+               GCO$            RCI$
                DIC$IS          DIC$MS          DIC$PS
+               DIC$SS          $DIVC           $DVC
                ADD$IS          ADD$MS          ADD$PS
+               ADD$SS          SUD$IS          SUD$MS
+               SUD$PS          SUD$SS          $ADD
```

The first line of the listing file shows the version of the librarian that was used and the current date and time. The second line prints the library file name and the date and time the library was created. Module names are not included in this example. Each line in the rest of the listing shows only the globals that appear in a particular module. If a module contains more global symbol names than can print on one line, a new line will be started with a plus (+) sign in column 1 to indicate continuation.

12.2.13 Merging Library Files

You can merge two or more library files under one file name by specifying in a single command line all the library files to be merged. The librarian does not delete the individual library files following the merge unless the output file name is identical to one of the input file names.

The command syntax is as follows:

```
*library-filespec=input-filespecs
```

where

library-filespec	represents the library file that will contain all the merged files. (If a library file already exists under this name, you must also indicate it in the input side of the command line so that it is included in the merge).
input-filespec	represents a library file to be merged.

Thus, the following command combines library files MAIN.OBJ, TRIG.OBJ, STP.OBJ, and BAC.OBJ under the existing library file name MAIN.OBJ; all files are on the default device DK:. Note that this replaces the old contents of MAIN.OBJ.

```
*MAIN=MAIN,TRIG,STP,BAC
```

The next command creates a library file named FORT.OBJ and merges existing library files A.OBJ, B.OBJ, and C.OBJ under the file name FORT.OBJ.

```
*FORT=A,B,C
```

NOTE

Library files that you combine using PIP are illegal as input to both the librarian and the linker.

12.2.14 Combining Library Option Functions

You can request two or more library functions in the same command line, with the exception of the /E option, which cannot be specified on the same command line with any other option. The librarian performs functions (and issues appropriate prompts) in the following order:

1. /C or //
2. /D
3. /G
4. /U
5. /R
6. Insertions
7. Listing

Here is an example that combines options:

```
*FILE,LP:=FILE/D,MODX,MODY/R
Module name? XYZ
Module name? A
Module name?
```

The librarian performs the functions in this example in order, as follows:

1. Deletes modules XYZ and A from the library file FILE.OBJ.
2. Replaces any duplicate of the modules in the file MODY.OBJ.
3. Inserts the modules in the file MODX.OBJ.
4. Lists the directory of FILE.OBJ on the line printer.

12.3 OPTION COMMANDS AND FUNCTIONS FOR MACRO LIBRARIES

The librarian lets you create macro libraries. A macro library works with the V03 MACRO-11 assembler to reduce macro search time.

The .MACRO directive produces the entries in the library directory (macro names). LIBR does not maintain a directory listing file for macro libraries; you can print the ASCII input file to list the macros in the library.

The default input and output file type for macro files is .MAC.

Be careful not to give the library file the same name as one of the input files. This deletes the input file when the library is created.

Table 12-2 summarizes the options you can use with macro libraries. The options are explained in detail in the following two sections.

Table 12-2 LIBR Macro Options

Options	Command Line	Section	Meaning
/C	any but last	12.3.1	Command continuation; allows you to type the input specification on more than one line.
/M[:n]	first	12.3.2	Macro; creates a macro library from the ASCII input file containing .MACRO directives.
//	first and last	12.3.1	Command continuation; allows you to type the input specification on more than one line.

12.3.1 Command Continuation Options (/C or //)

These options are the same for macro libraries as for object libraries. See Section 12.2.1.

12.3.2 Macro Option (/M[:n])

The /M[:n] option creates a macro library file from an ASCII input file that contains .MACRO directives. The optional argument, n, determines the amount of space to allocate for the macro name directory. Remember that n is interpreted as an octal number; you must follow n by a decimal point (n.) to indicate a decimal number. Each 64 macros occupy one block of library directory space. The default value for n is 128, enough space for 128 macros, which will use 2 blocks for the macro name table.

The command syntax is as follows:

```
*library-filespec=input-filespec/M[:n]
```

where

library-filespec represents the macro library to be created.

input-filespec	represents the ASCII input file that contains .MACRO definitions.
/M[:n]	is the macro option.

The continuation options (/C or //) are the only options you can use with the macro option.

The following example creates the macro library SYSMAC.SML from the ASCII input file SYSMAC.MAC. Both files are on device DK:.

```
*SYSMAC.SML=SYSMAC/M
```

CHAPTER 13
DUMP

DUMP is the RT-11 program that prints on the console or lineprinter, or writes to a file all or any part of a file in octal words, octal bytes, ASCII characters, and/or Radix-50 characters. DUMP is particularly useful for examining directories and files that contain binary data.

13.1 CALLING AND USING DUMP

To call the DUMP program from the system device, respond to the dot (.) printed by the keyboard monitor by typing:

R DUMP (RET)

The Command String Interpreter prints an asterisk at the left margin on the console terminal when it is ready to accept a command line. If you respond to the asterisk by typing only a carriage return, DUMP prints its current version number.

You can type CTRL/C to halt DUMP and return control to the monitor when DUMP is waiting for input from the console terminal. You must type two CTRL/Cs to abort DUMP at any other time. To restart DUMP, type R DUMP or REENTER and a carriage return in response to the monitor's dot. Chapter 6, Command String Interpreter, describes the general syntax of the command line that DUMP accepts. If you do not specify an output file, the listing prints on the line printer. If you do not specify a file type for an output file, the system uses .DMP.

13.2 DUMP OPTIONS

Table 13-1 summarizes the options that are valid for DUMP.

Table 13-1 DUMP Options

Option	Explanation
/B	Outputs octal bytes.
/E:n	Ends output at block number n, where n is an octal block number.
/G	Ignores input errors.
/N	Suppresses ASCII output.
/O:n	Outputs only block number n, where n is an octal block number. With the /O option, you can dump only one block for each command line.
/S:n	Starts output with block number n, where n is an octal block number. For random access devices, n cannot be greater than the number of blocks in the file.
/T	Defines a tape as non-RT-11 file-structured.
/W	Outputs octal words (the default mode).
/X	Outputs Radix-50 characters.

ASCII characters are always dumped unless you type /N.

If you specify an input file name, the block numbers (n) you supply are relative to the beginning of that file. If you do not specify a file name; that is, if you are dumping a device, the block numbers are the absolute (physical) block numbers on that device. Remember that the first block of any file or device is block 0.

DUMP handles operations that involve magtape and cassette differently from operations involving random access devices.

If you dump an RT-11 file-structured tape and specify only a device name in the input specification, DUMP reads only as far as the logical end-of-tape. Logical end-of-tape is indicated by an end-of-file label followed by two tape marks. For non-file-structured tape, logical end-of-tape is indicated by two consecutive tape marks. If you dump a cassette and specify only the device name in the input specification, the results are unpredictable. For magtape dumps, tape mark messages appear in the output listing as DUMP encounters them on the tape.

If you use /S:n with magtape, n can be any positive value. However, an error can occur if n is greater than the number of blocks written on the tape. For example, if a tape has 100 written blocks and n is 110, an error can occur if DUMP accesses past the 100th block. If you specify /E:n, DUMP reads the tape from its starting position (block 0, unless you specify otherwise) to block number n or to logical end-of-tape, whichever comes first.

13.3 EXAMPLES

This section includes sample DUMP commands and the listings they produce.

The following command string directs DUMP to print in octal words information contained in block 1 of the file DMPX.SAV stored on device DK:.

```
*DMPX.SAV/O:1

DMPX.SAV/O:1
BLOCK NUMBER 00001
000/ 012700 000030 000261 006101 106100 110024 012700 000206 *@...1.A.@...@...*
020/ 006301 001403 106100 103774 000766 000207 052140 023364 *A...@.\.V...@TT&*
040/ 022030 021424 015326 023747 000000 023747 006766 021401 *.S.#V.G'..G'V..*
060/ 050500 062745 177400 177001 051042 040505 020104 042515 *@QEE...~"READ ME*
100/ 021041 000000 000377 000001 000376 001000 000400 177777 *!"......~........*
120/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
140/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
160/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
200/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
220/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
240/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
260/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
300/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
320/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
340/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
360/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
400/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
420/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
440/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
460/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
500/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
520/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
540/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
560/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
600/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
620/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
640/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
660/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
700/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
720/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
740/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
760/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
```

In the printout above, the heading shows which block of the file follows. The numbers in the leftmost column indicate the byte offset from the beginning of the block. Remember that these are all octal values and that there are two bytes per word. The octal words that were dumped appear in the next eight columns. The rightmost column contains the ASCII equivalent of each octal word. DUMP substitutes a dot (.) for non-printing codes, such as those for control characters.

The next command dumps block 1 of file PIP.SAV. The /N option suppresses ASCII output.

```
*PIP.SAV/N/O:1

PIP.SAV/N/O:1
BLOCK NUMBER  00001
000/ 100101 000000 000000 000002 000001 100102 000000 000000
020/ 000001 000002 100103 000000 000000 000000 000000 000104
040/ 000000 177352 002000 000004 100107 000000 000000 000000
060/ 000000 100113 000000 000000 006002 000020 100115 000000
100/ 000000 002000 000040 100116 000000 000000 000500 000200
120/ 100117 000000 000000 000300 000400 100120 000000 000000
140/ 002000 001000 100121 000000 000000 000000 000000 000122
160/ 000000 177602 001164 002000 100123 000000 000000 000000
200/ 000000 100124 000000 000000 000000 000000 100125 000000
220/ 000000 000020 004000 100127 000000 000000 000000 000000
240/ 100130 000000 000000 000000 000000 100131 000000 000000
260/ 000000 000000 000000 100115 000000 000000 002600 000100
300/ 004000 000001 000000 000100 000000 000000 000000 000000
320/ 000000 000000 000000 000000 000000 000000 000000 000000
340/ 000000 000000 000000 000000 000000 000000 000000 000000
360/ 000000 000000 000000 000000 000000 000000 000000 000000
400/ 000000 000000 000000 000000 000000 000000 000000 000000
420/ 000000 000000 000000 000000 000000 000000 000000 000000
440/ 000000 000000 000000 000000 000000 000000 000000 000000
460/ 000000 000000 000000 000000 000000 000000 000000 000000
500/ 000000 000000 000000 000000 000000 000000 000000 000000
520/ 000000 000000 000000 000000 000000 000000 000000 000000
540/ 000000 000000 000000 000000 000000 000000 000000 000000
560/ 000000 000000 000000 000000 000000 000000 000000 000000
600/ 000000 000000 000000 000000 000000 000000 000000 000000
620/ 000000 000000 000000 000000 000000 000000 000000 000000
640/ 000000 000000 000000 000000 000000 000000 000000 000000
660/ 000000 000000 000000 000000 000000 000000 000000 000000
700/ 000000 000000 000000 000000 000000 000000 000000 000000
720/ 000000 000000 000000 000000 000000 000000 000000 000000
740/ 000000 000000 000000 000000 003054 002543 002510 002562
760/ 002314 002407 002426 002342 002446 002614 002676 002177
```

The following command dumps block 1 of SYSMAC.MAC in octal bytes. ASCII equivalents appear underneath each byte.

```
*SYSMAC.MAC/B/O:1

SYSMAC.MAC/B/O:1
BLOCK NUMBER  00001
000/ 040 124 117 040 124 110 105 123 105 040 114 111 103 105 116 123
       T   O       T   H   E   S   E       L   I   C   E   N   S
020/ 105 040 124 105 122 115 123 056 040 124 111 124 114 105 040 124
       E       T   E   R   M   S   .       T   I   T   L   E       T
040/ 117 040 101 116 104 040 117 127 116 105 122 123 110 111 120 040
       O       A   N   D       O   W   N   E   R   S   H   I   P
060/ 117 106 040 124 110 105 040 015 012 073 040 123 117 106 124 127
       O   F       T   H   E       .   .   ;       S   O   F   T   W
100/ 101 122 105 040 123 110 101 114 114 040 101 124 040 101 114 114
       A   R   E       S   H   A   L   L       A   T       A   L   L
```

```
120/ 040 124 111 115 105 123 040 122 105 115 101 111 116 040 111 116
       T   I   M   E   S       R   E   M   A   I   N       I   N
140/ 040 104 111 107 111 124 101 114 056 015 012 073 015 012 073 040
       D   I   G   I   T   A   L   .   .   .   ;   .   .   ;
160/ 124 110 105 040 111 116 106 117 122 115 101 124 111 117 116 040
       T   H   E       I   N   F   O   R   M   A   T   I   O   N
200/ 111 116 040 125 110 111 123 040 123 117 106 124 127 101 122 105
       I   N       T   H   I   S       S   O   F   T   W   A   R   E
220/ 040 111 123 040 123 125 102 112 105 103 124 040 124 117 015 012
           I   S       S   U   B   J   E   C   T       T   O   .   .
240/ 073 040 103 110 101 116 107 105 040 127 111 124 110 117 125 124
       ;       C   H   A   N   G   E       W   I   T   H   O   U   T
260/ 040 116 117 124 111 103 105 040 101 116 104 040 123 110 117 125
           N   O   T   I   C   E       A   N   D       S   H   O   U
300/ 114 104 040 116 117 124 040 102 105 040 103 117 116 123 124 122
       L   D       N   O   T       B   E       C   O   N   S   T   R
320/ 125 105 104 015 012 073 040 101 123 040 101 040 103 117 115 115
       U   E   D   .   .   ;       A   S       A       C   O   M   M
340/ 111 124 115 105 116 124 040 102 131 040 104 111 107 111 124 101
       I   T   M   E   N   T       B   Y       D   I   G   I   T   A
360/ 114 040 105 121 125 111 120 115 105 116 124 040 103 117 122 120
       L       E   Q   U   I   P   M   E   N   T       C   O   R   P
400/ 117 122 101 124 111 117 116 056 015 012 073 015 012 073 040 104
       O   R   A   T   I   O   N   .   .   .   ;   .   .   ;       D
420/ 111 107 111 124 101 114 040 101 123 123 125 115 105 123 040 116
       I   G   I   T   A   L       A   S   S   U   M   E   S       N
440/ 117 040 122 105 123 120 117 116 123 111 102 111 114 111 124 131
       O       R   E   S   P   O   N   S   I   B   I   L   I   T   Y
460/ 040 106 117 122 040 124 110 105 040 125 123 105 015 012 073 040
           F   O   R       T   H   E       U   S   E   .   .   ;
500/ 117 122 040 122 105 114 111 101 102 111 114 111 124 131 040 117
       O   R       R   E   L   I   A   B   I   L   I   T   Y       O
520/ 106 040 111 124 123 040 123 117 106 124 127 101 122 105 040 117
       F       I   T   S       S   O   F   T   W   A   R   E       O
540/ 116 040 105 121 125 111 120 115 105 116 124 015 012 073 040 127
       N       E   Q   U   I   P   M   E   N   T   .   .   ;       W
560/ 110 111 103 110 040 111 123 040 116 117 124 040 123 125 120 120
       H   I   C   H       I   S       N   O   T       S   U   P   P
600/ 114 111 105 104 040 102 131 040 104 111 107 111 124 101 114 056
       L   I   E   D       B   Y       D   I   G   I   T   A   L   .
620/ 015 012 073 015 012 073 040 105 106 054 112 104 054 114 120 054
       .   .   ;   .   .   ;       E   F   ,   J   D   ,   L   P   ,
640/ 102 103 054 104 126 054 103 122 054 110 112 015 012 014 056 115
       B   C   ,   D   V   ,   C   R   ,   H   J   .   .   .   .   M
660/ 101 103 122 117 040 056 056 126 061 056 056 015 012 056 115 103
       A   C   R   O       .   .   V   1   .   .   .   .   .   M   C
700/ 101 114 114 011 056 056 056 103 115 060 054 056 056 056 103 115
       A   L   L   .   .   .   .   C   M   0   ,   .   .   .   C   M
720/ 061 054 056 056 056 103 115 062 054 056 056 056 103 115 063 054
       1   ,   .   .   .   C   M   2   ,   .   .   .   C   M   3   ,
740/ 056 056 056 103 115 064 054 056 056 056 103 115 065 054 056 056
       .   .   .   C   M   4   ,   .   .   .   C   M   5   ,   .   .
760/ 056 103 115 066 015 012 056 056 056 126 061 075 061 056 015 012
       .   C   M   6   .   .   .   .   .   V   1   =   1   .   .   .
```

The last example shows block 6 (the directory) of device RK0:. The output is in octal words with Radix-50 equivalents below each word.

```
*RK0:/N/X/O:6

RK0:/N/X/O:6
BLOCK NUMBER  00006
000/ 000020 000004 000004 000000 000046 002000 071105 055202
       P      D      D        0      8      YX     RKM    NSJ
020/ 075273 000130 000015 010405 002000 071105 054162 075273
     SYS      BH     M     B,7     YX     RKM    NFB    SYS
040/ 000141 000015 010405 002000 071105 055515 075273 000150
      BQ       M     B,7     YX     RKM    NXM    SYS     BX
```

```
060/ 000015 010405 002000 015425 055202 075273 000132 000015
       M     B,7      YX     DMM     NSJ     SYS      BJ      M
100/ 010405 002000 015425 054162 075273 000143 000015 010405
     B,7      YX     DMM     NFB     SYS      BS      M     B,7
120/ 002000 015425 055515 075273 000152 000015 010405 002000
       YX     DMM     NXM     SYS      BZ      M     B,7      YX
140/ 016315 055202 075273 000130 000015 010405 002000 016315
     DXM     NSJ     SYS      BH      M     B,7      YX     DXM
160/ 054162 075273 000141 000015 010405 002000 016315 055515
     NFB     SYS      BQ      M     B,7      YX     DXM     NXM
200/ 075273 000141 000015 010405 002000 016055 055202 075273
     SYS      BQ      M     B,7      YX     DTM     NSJ     SYS
220/ 000130 000015 010405 002000 016055 054162 075273 000141
      BH      M     B,7      YX     DTM     NFB     SYS      BQ
240/ 000015 010405 002000 016055 055515 075273 000150 000015
       M     B,7      YX     DTM     NXM     SYS      BX      M
260/ 010405 002000 016005 055202 075273 000130 000015 010405
     B,7      YX     DSM     NSJ     SYS      BH      M     B,7
300/ 002000 016005 054162 075273 000141 000015 010405 002000
       YX     DSM     NFB     SYS      BQ      M     B,7      YX
320/ 016005 055515 075273 000150 000015 010405 002000 015615
     DSM     NXM     SYS      BX      M     B,7      YX     DPM
340/ 055202 075273 000130 000015 010405 002000 015615 054162
     NSJ     SYS      BH      M     B,7      YX     DPM     NFB
360/ 075273 000141 000015 010405 002000 015615 055515 075273
     SYS      BQ      M     B,7      YX     DPM     NXM     SYS
400/ 000150 000015 010405 002000 070575 055202 075273 000130
      BX      M     B,7      YX     RFM     NSJ     SYS      BH
420/ 000015 010405 002000 070575 054162 075273 000141 000015
       M     B,7      YX     RFM     NFB     SYS      BQ      M
440/ 010405 002000 070575 055515 075273 000150 000015 010405
     B,7      YX     RFM     NXM     SYS      BX      M     B,7
460/ 002000 071105 056573 075273 000123 000015 010405 002000
       YX     RKM     N8K     SYS      BC      M     B,7      YX
500/ 016315 056573 075273 000123 000015 010405 002000 016040
     DXM     N8K     SYS      BC      M     B,7      YX     DT
520/ 000000 075273 000002 000015 010405 002000 015600 000000
             SYS       B       M     B,7      YX     DP
540/ 075273 000002 000015 010405 002000 016300 000000 075273
     SYS       B       M     B,7      YX     DX             SYS
560/ 000003 000015 010405 002000 070560 000000 075273 000002
       C       M     B,7      YX     RF             SYS       B
600/ 000015 010405 002000 071070 000000 075273 000002 000015
       M     B,7      YX     RK             SYS       B       M
620/ 010405 002000 015410 000000 075273 000004 000015 010405
     B,7      YX     DM             SYS       D       M     B,7
640/ 002000 015770 000000 075273 000002 000015 010405 002000
       YX     DS             SYS       B       M     B,7      YX
660/ 100040 000000 075273 000002 000015 010405 002000 046600
      TT             SYS       B       M     B,7      YX     LP
700/ 000000 075273 000002 000015 010405 002000 012620 000000
             SYS       B       M     B,7      YX     CR
720/ 075273 000003 000015 010405 002000 052140 000000 075273
     SYS       C       M     B,7      YX     MT             SYS
740/ 000010 000015 010405 002000 051510 000000 075273 000011
       H       M     B,7      YX     MM             SYS       I
760/ 000015 010405 002000 054540 000000 075273 000002 000015
       M     B,7      YX     NL             SYS       B       M
```

CHAPTER 14
FILEX

The file exchange program (FILEX) is a general file transfer program that converts files from one format to another so that you can use them with various operating systems. You can initiate transfers between any block-replaceable RT-11 directory-structured device and any device listed in Table 14-1.

Table 14-1 Legal FILEX Devices

Device	Valid as Input	Valid as Output
PDP-11 DOS/BATCH DECtape	X	X
DOS/BATCH Disk	X	
RSTS DECtape	X	X
DECsystem-10 DECtape	X	
Interchange Diskette	X	X

FILEX does not support magtape or cassette in any operation.

Section 4.2 of this manual describes how to use wildcards. You can use wildcards in the FILEX command string. However, you can not use embedded wildcards in any file name or file type. For example, the following line represents a valid file specification.

```
**.MAC
```

The next line is an illegal file specification for FILEX.

```
*T%ST.MAC
```

14.1 FILE FORMATS

FILEX can transfer files created by four different operating systems: RT-11, DECsystem-10, universal interchange format (IBM) and DOS/BATCH (PDP-11 Disk Operating System). You can use the following three data formats in a transfer: ASCII, image, and packed image. ASCII files conform to the American Standard Code for Information Interchange in which each character is represented by a 7-bit code. In ASCII mode, FILEX deletes null and rubout characters, as well as parity bits.

Because the file structure and data formats for each system vary, options are needed in the command line to indicate the file structures and the data formats involved in the transfer. These options are discussed in Section 14.3. FILEX assumes that all devices are RT-11 structured. You can use options from Table 14-2 to indicate otherwise.

14.2 CALLING AND USING FILEX

To call FILEX from the system device, respond to the dot (.) printed by the keyboard monitor by typing:

R FILEX (RET)

The Command String Interpreter prints an asterisk at the left margin of the terminal and waits for you to enter a command. If you enter only a carriage return at this point, the current version number of FILEX prints on the terminal.

Type two CTRL/Cs to halt FILEX at any time (or a single CTRL/C to halt FILEX when it is waiting for console terminal input) and return control to the monitor. To restart FILEX, type R FILEX or REENTER in response to the monitor's dot.

14.3 FILEX OPTIONS

Table 14-2 lists the options that initiate various FILEX operations. The table is divided into three sections: transfer options, operation options and file structure options. Transfer options direct FILEX to copy data in a certain mode. The three transfer modes are: ASCII, image, and packed image. Operation options perform another function in addition to the data transfer. These additional functions include deleting files, producing directory listings and zeroing device directories. File structure options indicate the formats of devices that are involved in a transfer. These formats are DOS/BATCH or RSTS, DECsystem-10, and interchange. FILEX accepts one transfer option and one operation option in a single command. You can specify one device option for each file involved in the transfer. The device options (/S, /T, and /U) must appear following the device and file name to which they apply; other options can appear anywhere in the command line. These options are explained in more detail in the following sections.

14.3.1 Transferring Files Between RT-11 and DOS/BATCH (or RSTS)

You can transfer files between block-replaceable devices used by RT-11 and the PDP-11 DOS/BATCH system. Input from DOS/BATCH can be either disk or DECtape. You can use both linked and contiguous files.

If the input device is a DOS/BATCH disk, you should specify a DOS/BATCH user identification code (UIC). The UIC is of the form [nnn,nnn], where nnn represents an octal integer less than or equal to 377. The first part of the code represents a user-group number; the second is the individual user number. The initial default value is [1,1]. The UIC you supply will be the default for all future transfers. If you do not specify a UIC, FILEX will use the current default UIC. Note that the square brackets [ ] are part of the UIC; you must type them if you specify a UIC.

Output to DOS/BATCH is limited to DECtape only. You do not need a UIC in a command line when you are accessing only DECtape. Individual users do not "own" files on DECtape under DOS. However, no error occurs if you do use a UIC. DECtape used under the RSTS system is legal as both input and output, since its format is identical to DOS/BATCH DECtape. You can use any valid RT-11 file storage device for either input or output in the transfer. The RT-11 device DK: is assumed if you do not indicate a device.

An RT-11 DECtape can hold more information than a DOS/BATCH or RSTS DECtape. Be careful when you copy files from a full RT-11 tape to a DOS DECtape. Some information might not transfer. In this case, an error message prints and the transfer does not complete.

When a transfer from an RT-11 device to a DOS DECtape occurs, the block size of the file can increase. However, if the file is later transferred back to an RT-11 device, the block size does not decrease.

Table 14-2 FILEX Options

Transfer Options	Explanation
/A	Indicates a character-by-character ASCII transfer in which FILEX deletes rubouts and nulls. If you use /U with /A, FILEX ignores all sector boundaries on the diskette. If you use /T with /A, FILEX assumes that each PDP-10 36-bit word contains five 7-bit ASCII bytes. If you use /U with /A, FILEX assumes that records are to be terminated by a line feed, vertical tab, or form feed. The transfer terminates when a CTRL/Z is encountered. (This feature is included for compatibility with RSTS.) FILEX does not transfer the CTRL/Z.
/I	Performs an image mode transfer. If the input is DOS/BATCH, RSTS, interchange diskette, or RT-11, the transfer is word-for-word. If the input is from DECsystem-10, /I indicates that the file resembles a file created on DECsystem-10 by MACY11, MACX11, or LNKX11 with the /I option. In this case, each PDP-10 36-bit word will contain one PDP-11 8-bit byte in its low-order bits. If input or output is an interchange diskette, FILEX reads and writes four diskette sectors for each RT-11 block.
/P	Performs a packed image mode transfer. If the input is DOS/BATCH, RSTS, or RT-11, the transfer is word-for-word. If the input is from DECsystem-10, /P indicates that the file resembles a file created on DECsystem-10 by MACY11, MACX11, or LNKX11 with the /P option. In this case, each PDP-10 36-bit word will contain four PDP-11 8-bit bytes aligned on bits 0, 8, 18, and 26. This is the default mode. If the input is interchange diskette, the data is assumed to be EBCDIC. If the output is interchange diskette, FILEX writes the data as EBCDIC.
Operation Options	**Explanation**
/D	Deletes the file you specify from the device directory. This option is valid only for DOS/BATCH, RSTS DECtape, and interchange diskette.
/F	Produces a brief listing of the device directory on the terminal. It lists only file names and file types.
/L	Produces a complete listing of the device directory on the console terminal, including file names, block lengths, and creation dates.
/Y	Suppresses the dev:/ZERO ARE YOU SURE? message.
/Z	Zeroes the directory of the device you specify in the proper format. This option is valid only for DOS/BATCH, RSTS DECtape, and interchange diskette.
File Structure Options	**Explanation**
/S	Indicates that the device is a legal DOS/BATCH or RSTS block-replaceable device.
/T	Indicates that the device is a legal DECsystem-10 DECtape.
/U[:n.]	Indicates that the device is an interchange diskette; n. represents the length of each output record, in characters; n. is a decimal integer in the range 1-128. The default value is 80; n. is not valid with an input file specification, or with /A or /I.

To transfer a file from a legal DOS/BATCH block-replaceable device or RSTS DECtape to a legal RT-11 device, use this command syntax:

*output-filespec=input-filespec/S[/option]

where

output-filespec	represents any valid RT-11 device, file name, and file type (if the device is not file structured, you can omit the file name and file type).
input-filespec	represents the DOS/BATCH or RSTS device, UIC, file name and file type to be transferred. See Table 14-1 for a list of valid devices.
/S	is the option from Table 14-2 that designates a DOS/BATCH or RSTS block-replaceable device. This option must be included in the command line.
/option	is one of the three transfer options from Table 14-2.

To transfer files from an RT-11 storage device to a DOS/BATCH or RSTS DECtape, use this command syntax:

*DTn:output-filename/S[/option]=input-filespec

where

DTn:output-filename	represents the file name and file type of the file to be created, as well as the DOS/BATCH or RSTS DECtape on which to store the file.
input-filespec	represents the device, file name, and file type of the RT-11 file to be transferred.
/S	is the option from Table 14-2 that designates a DOS/BATCH or RSTS DECtape. This option must be included in the command line.
/option	is one of the three transfer options from Table 14-2.

The following examples illustrate the use of the /S option.

The following command instructs FILEX to transfer a file called SORT.ABC from the RT-11 default device DK: to a DOS/BATCH or RSTS format DECtape on unit DT2. The transfer is done in image mode.

```
*DT2:SORT.ABC/S=SORT.ABC/I
```

The next command allows a file to be transferred from DOS/BATCH (or RSTS) DECtape to the papertape punch under RT-11. The transfer is done in ASCII mode.

```
*PC:=DT2:FIL.TYP/S/A
```

The next command causes the file MACR1.MAC from the DOS/BATCH disk on unit 1, which is stored under the UIC [1,2], to be transferred to the RT-11 device DK:. [1,2] becomes the default UIC for any further DOS/BATCH operations.

```
*DK:*.*=RK1:[1,2]MACR1.MAC/S
```

14.3.2 Transferring Files Between RT-11 and Interchange Diskette

You can transfer files between block-replaceable devices used by RT-11 and interchange format (proposed ANSI format) diskettes. Files are transferred in one of the following three formats: ASCII, image, and packed image (EBCDIC) mode.

A universal diskette consists of 77 tracks (some of which are reserved), each containing 26 sectors numbered from 1 to 26. A sector contains one record of 128 or fewer characters. A record must begin on a sector boundary on an interchange diskette in packed image mode. There must be only one record per sector. If a record does not fill a sector, the remainder is filled with blanks. Since packed image (EBCDIC) mode is inefficient and wastes space, it is only recommended to read or write diskettes that must be compatible with IBM 3741 format.

Image mode provides an exact copy of a file. Nulls, rubouts, and parity are preserved in a transfer. ASCII and image mode perform similar functions; however, for most operations, you should probably use ASCII. Use image mode to transfer data when the parity bit or nulls are significant (i.e., when you are not transferring ASCII data).

Packed image (EBCDIC) mode is generally compatible with IBM 3741 format. (FILEX does not support error mapping of bad sectors and multi-volume files.) Packed image (EBCDIC) is the default mode, so you must use one of the options from Table 14-2 to specify ASCII or image mode. All records of a file must be the same size. You indicate this with the /U:n. option.

NOTE

File types are not normally recognized in interchange format; instead, a single 8-character file name is used. However, in order to provide uniformity throughout RT-11, FILEX has been designed to accept a 6-character file name with a 2-character file type. If you transfer a file from RT-11 to interchange diskette, any 3-character file type is truncated to two characters.

To transfer files from RT-11 format to interchange format, use this command syntax:

```
*output-filespec/U[:n.] [/option]=input-filespec
```

where

output-filespec	represents the device, file name, and file type of the interchange file to be created.
/U[:n.]	is the option from Table 14-2 that designates an interchange diskette. This option must be included in the command line; n. represents the length of each output record, in characters; $1 \leqslant n \leqslant 128$ (default is 80).
/option	is one of the three transfer options from Table 14-2.
input-filespec	represents the device, file name, and file type of the RT-11 file to be transferred.

To transfer files from interchange diskette to RT-11 format, use this command syntax:

```
*output-filespec=input-filespec/U[/option]
```

where

output-filespec	represents the device, file name, and file type of the RT-11 file to be created.
input-filespec	represents the device, file name, and file type of the interchange file to be transferred.
/U	is the option from Table 14-2 that designates an interchange diskette. This option must be included in the command line.
/option	is one of the three transfer options from Table 14-2.

The following command transfers the file IVAN.CAT from RT-11 RK05 unit 2 to the diskette on unit 1. The transfer is done in exact image mode (indicated by /I), ignoring all sector boundaries.

```
*DX1:IVAN.CAT/U/I=RK2:IVAN.CAT
```

The next command instructs FILEX to transfer the file BENMAR.FRM from the RT-11 disk unit 2 to the diskette on unit 0, and rename it KENJOS.JO. The /U option indicates that the format is to be changed from ASCII to the interchange format. There will be one record per sector of 128 or fewer characters. If there are fewer than 128 characters, the remainder of the sector will be filled with spaces.

```
*DX0:KENJOS.JO/U=RK2:BENMAR.FRM
```

The next command transfers the file TYPE.SET from RT-11 diskette unit 0 to the interchange diskette on unit 2. The exchange converts ASCII to interchange format putting a maximum of 7 (indicated by :7.) characters into each sector until the entire record has been transferred. Records in excess of seven characters will be broken up and placed in succeeding sectors on the diskette. New records always begin on a sector boundary; carriage returns and line feeds are discarded. However, if you use /A or /I, FILEX ignores boundary limits and preserves carriage returns and line feeds.

```
*DX2:TYPE.SE/U:7.=RX0:TYPE.SET
```

File TYPE.SET before transfer:

```
ABCDEFGHIJKLMN
```

File TYPE.SET after transfer:

```
ABCDEFG – (spaces up to 128 characters) Sector 1
HIJKLMN – (spaces up to 128 characters) Sector 2
```

The next command copies file IVAN.CA from the interchange diskette on unit 1 to the RT-11 line printer, treating the input as ASCII characters. Note that once a record has been divided into sectors, it cannot be transferred back to its original large size.

```
*LP:=DX1:IVAN.CA/U/A
```

14.3.3 Transferring Files to RT-11 from DECsystem-10

Files can not be transferred to RT-11 devices from a DECsystem-10 DECtape when a foreground job is running. This restriction is due to the fact that when FILEX reads DECsystem-10 files, it accesses the DECtape control registers directly instead of using the RT-11 DECtape control handler. Output can be to any valid RT-11 device. DECsystem-10 DECtape is the only valid input device. To transfer files from DECsystem-10 format to RT-11 format, use this command syntax:

*output-filespec=input-filespec/T[/option]

where

output-filespec	represents any valid RT-11 device, file name, and file type (if the device is not file-structured, you can omit the file name and file type).
input-filespec	represents the DECtape unit, file name, and file type of the DECsystem-10 file to be transferred.
/T	is the option from Table 14-2 that signifies a DECsystem-10 DECtape. When you use /T, and especially when you also use /A, the system clock loses time. Examine the time and reset it if necessary with the TIME command.
/option	is one of the three transfer options from Table 14-2.

You can not convert RT-11 files to DECsystem-10 format directly. However, there is a two-step procedure for doing this. First, run RT-11 FILEX and convert the files to DOS formatted DECtape. Then run DECsystem-10 FILEX to read the DOS DECtape.

The following command converts the ASCII file STAND.LIS from DECsystem-10 ASCII format to RT-11 ASCII format and stores it under RT-11 on DECtape 2 as STAND.LIS.

```
*DT2:STAND.LIS=DT1:STAND.LIS/T/A
```

Transfers from DECsystem-10 DECtape to RT-11 DECtape can cause an <UNUSED> block to appear after the file on the RT-11 device. This is a result of the method by which RT-11 handles the increased amount of information on a DECsystem-10 DECtape.

The next command indicates that all files on DECsystem-10 DECtape 0 with the file type .LIS are to be transferred to the RT-11 system device using the same file name and a file type of .NEW. The /P option is the assumed transfer mode.

```
*SY:*.NEW=DT0:*.LIS/T
```

14.3.4 Listing Directories

You can list a directory of any of the block-replaceable devices used in a FILEX transfer. The directory listing prints on the console terminal. The command syntax is:

*device:/L/option

where

device	represents the block-replaceable device. These are the valid device types:		
	DOS/BATCH, RSTS	DTn:	or any disk
	DECsystem-10	DTn:	
	interchange diskette	DXn:	
/L	is the listing option from Table 14-2. You can substitute /F if you want a brief listing of file names only.		

/option	is /S, /T, or /U[:n.]. These are the valid format and option combinations:	
	DOS/BATCH, RSTS	/S
	DECsystem-10	/T
	interchange diskette	/U

The following example shows the complete disk directory for UIC[1,7] of the device RK1:. The letter C following the file size on a DOS/BATCH or RSTS directory listing indicates that the file is a contiguous file.

```
*RK1:/L/SY[1,7]
BADB       .SYS       1          22-JUL-74
MONLIB     .CIL     175C         22-JUL-74
DU11       .PAL      45          24-JUL-74
VERIFY     .LDA      67C         22-JUL-74
CILUS      .LDA      39          22-JUL-74
```

The next command lists all files with the file type .PAL that are stored on DECtape unit 1.

```
*DT1:*.PAL/L/S
```

The next command produces a brief directory listing of the interchange diskette on unit 0, giving file names only.

```
*DX0:/U/F
```

The following command lists all files on DECsystem-10 formatted DECtape unit 1, regardless of file name or file type; a brief directory is requested (/F) in which only file names print.

```
*DT1:*.*/F/T
```

14.3.5 Deleting Files From DOS/BATCH (RSTS) DECtapes and Interchange Diskettes

Use FILEX to delete files from DOS/BATCH and RSTS formatted DECtapes, and from interchange diskettes.

To delete files, use this command syntax:

```
*filespec/D/option
```

where

filespec	represents the device, file name and file type of the file to be deleted.
/D	is the delete option from Table 14-2.
/option	can be either /S, for DOS/BATCH and RSTS block-replaceable devices, or /U, for interchange diskettes.

The following command deletes all files with the file type .PAL on DECtape unit 0.

```
*DT0:*.PAL/D/S
```

The next command deletes the file TABLE.OBJ from the DECtape on unit 2.

```
*DT2:TABLE.OBJ/D/S
```

The next command deletes all files with an .RN file type from the interchange diskette on unit 0.

```
*DX0:*.RN/D/U
```

You can also use FILEX to initialize the directories of DOS/BATCH and RSTS DECtapes, and interchange diskettes. Use this command syntax:

*device:/Z/option[/Y]

where

device	represents the DOS/BATCH or RSTS DECtape, or the interchange diskette to be zeroed.
/Z	is the zero option from Table 14-2.
/option	can be either /S, for DOS/BATCH and RSTS DECtapes, or /U, for interchange diskettes.
/Y	inhibits the FILEX verification message.

The following command directs FILEX to initialize the directory of the interchange diskette on unit 0.

```
*DX0:/Z/U
DX0:/Zero are you sure?
```

Respond with a Y for initialization to begin. Any other response aborts the command.

The next command initializes the DECtape on unit 1 in DOS/BATCH (RSTS) format. Note that by using the /Y option you suppress the verification message.

```
*DT1:/Z/S/Y
```

NOTE

An initialized universal diskette's directory has a single file entry, DATA, that reserves the entire diskette. You must delete this file before you can write any new files on this diskette. This arrangement is necessary for IBM compatibility. Do this by using the following command:

*DX0:DATA/D/U

CHAPTER 15
SOURCE COMPARE (SRCCOM)

The RT-11 source compare program (SRCCOM) compares two ASCII files and lists the differences between them. SRCCOM can either print the results or store them in a file. SRCCOM is particularly useful when you need to compare two similar versions of a source program. A file comparison listing highlights the changes made to a program during an editing session.

15.1 CALLING AND USING SRCCOM

To call SRCCOM from the system device, respond to the dot (.) printed by the keyboard monitor by typing:

R SRCCOM (RET)

The Command String Interpreter prints an asterisk at the left margin of the terminal and waits for you to enter a command string. If you respond to the asterisk by entering only a carriage return, SRCCOM prints its current version number. The syntax of the command is:

[output-filespec=] input-filespec1,input-filespec2[/option . . .]

where

output-filespec	represents the destination device or file for the listing of differences.
input-filespec1	represents the first file to be compared.
input-filespec2	represents the second file to be compared.
option	is one of the options from Table 15-1.

The console terminal is the default output device. The default file type for input files is .MAC. SRCCOM assigns .DIF as the default file type for output files.

You can type CTRL/C to halt SRCCOM and return control to the monitor when SRCCOM is waiting for input from the console terminal. You must type two CTRL/Cs to abort SRCCOM at any other time. To restart SRCCOM, type R SRCCOM or REENTER and a carriage return in response to the monitor's dot.

SRCCOM examines the two source files line by line, looking for groups of lines that match. When SRCCOM finds a mismatch, it lists the lines from each file that are different. SRCCOM continues to list the differences until a specific number of lines from the first file match the second file. The specific number of lines that constitutes a match is a variable that you can set with the /L:n option.

15.2 SRCCOM OPTIONS

Table 15-1 summarizes the operations you can perform with SRCCOM. You can place these options anywhere in the command string, but it is conventional to place them at the end of the command line.

Table 15-1 SRCCOM Options

Option	Explanation
/B	Compares blank lines; normally, SRCCOM ignores blank lines.
/C	Ignores comments (all text on a line preceded by a semicolon) and spacing (spaces and tabs). A line consisting entirely of a comment is still included in the line count.
/F	Includes form feeds in the output listing; SRCCOM normally compares form feeds, but does not include them in the output listing.
/H	Types on the console terminal a list of options available; this is the "help" text.
/L:n	Specifies the number of lines that determines a match; n is an octal integer in the range 1-310. The default value for n is 3.
/S	Ignores spaces and tabs.

15.3 SRCCOM OUTPUT FORMAT

This section describes the SRCCOM output listing format and explains how to interpret it.

15.3.1 Sample Text

It will be helpful first to look at a sample text file, DEMO.BAK:

```
        FILE1
HERE'S A BOTTLE AND AN HONEST FRIEND!
  WHAT WAD YE WISH FOR MAIR, MAN?
WHA KENS, BEFORE HIS LIFE MAY END,
  WHAT HIS SHAME MAY BE O' CARE, MAN?
THEN CATCH THE MOMENTS AS THEY FLY,
  AND USE THEM AS YE OUGHT, MAN: --
BELIEVE ME, HAPPINESS IS SLY,
  AND COMES NOT AY WHEN SOUGHT, MAN.

                        --SCOTTISH SONG
```

This file contains two typing errors. In the fourth line of the song, "shame" should be "share". In the seventh line, "sly" should be "shy". Here is a file called DEMO.TXT that has the correct text:

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

                        --SCOTTISH SONG
```

15.3.2 Sample Output Listing

SRCCOM lists the differences between the two files. The example below compares the original file, DEMO.BAK, to its edited version, DEMO.TXT:

```
*DEMO.BAK,DEMO.TXT/L:1
1)1                FILE1
2)1                FILE2

1)1          WHAT HIS SHAME MAY BE O' CARE, MAN?
1)         THEN CATCH THE MOMENTS AS THEY FLY,
****
2)1          WHAT HIS SHARE MAY BE O' CARE, MAN?
2)         THEN CATCH THE MOMENTS AS THEY FLY,
**********
1)1        BELIEVE ME, HAPPINESS IS SLY,
1)           AND COMES NOT AY WHEN SOUGHT, MAN.
****
2)1        BELIEVE ME, HAPPINESS IS SHY,
2)           AND COMES NOT AY WHEN SOUGHT, MAN.
**********

%FILES ARE DIFFERENT
```

SRCCOM always prints the first line of each file as identification:

```
1)1                FILE1
2)1                FILE2
```

The numbers at the left margin have the form n)m, where n represents the source file (either 1 or 2) and m represents the page of that file on which the specific line is located.

SRCCOM next prints a blank line and then lists the differences between the two files. The /L:n option was used in this example to set to 1 the number of lines that must agree to constitute a match.

The first three lines of the song are the same in both files, so they do not appear in the listing. The fourth line contains the first discrepancy. SRCCOM prints the fourth line from the first file, followed by the next matching line as a reference.

```
1)1          WHAT HIS SHAME MAY BE O' CARE, MAN?
1)         THEN CATCH THE MOMENTS AS THEY FLY,
****
```

The four asterisks terminate the differences section from the first file.

SRCCOM then prints the fourth line from the second file, again followed by the next matching line as a reference:

```
2)1          WHAT HIS SHARE MAY BE O' CARE, MAN?
2)         THEN CATCH THE MOMENTS AS THEY FLY,
**********
```

The ten asterisks terminate the listing for a particular difference section.

SRCCOM scans the remaining lines in the files in the same manner. When it reaches the end of each file, it prints the %FILES ARE DIFFERENT message on the terminal.

The second example is slightly different. The default value for the /L:n option sets to 3 the number of lines that must agree to constitute a match. The output listing is directed to the file DIFF.TXT on device DK:.

```
*DIFF.TXT=DEMO.BAK,DEMO.TXT

%FILES ARE DIFFERENT
```

The monitor TYPE command lists the information contained in the output file:

```
.TYPE DIFF.TXT

1)1                FILE1
2)1                FILE2

1)1          WHAT HIS SHAME MAY BE O' CARE, MAN?
1)         THEN CATCH THE MOMENTS AS THEY FLY,
1)           AND USE THEM AS YE OUGHT, MAN:--
1)         BELIEVE ME, HAPPINESS IS SLY,
1)           AND COMES NOT AY WHEN SOUGHT, MAN.
1)
1)                            --SCOTTISH SONG
1)
****
2)1          WHAT HIS SHARE MAY BE O' CARE, MAN?
2)         THEN CATCH THE MOMENTS AS THEY FLY,
2)           AND USE THEM AS YE OUGHT, MAN:--
2)         BELIEVE ME, HAPPINESS IS SHY,
2)           AND COMES NOT AY WHEN SOUGHT, MAN.
2)
2)                            --SCOTTISH SONG
2)
**********
```

As in the first example, SRCCOM prints the first line of each file:

```
1)1                FILE1
2)1                FILE2
```

The first three lines of each file are identical and, therefore, constitute a match. Again, the fourth lines differ. SRCCOM prints the fourth line of the first file, followed by the next matching line:

```
1)1          WHAT HIS SHAME MAY BE O' CARE, MAN?
1)         THEN CATCH THE MOMENTS AS THEY FLY,
```

However, SRCCOM did not find a match (three identical lines) before it encountered the next difference. So, the second matching line prints, followed by the next differing line from the first file:

```
1)           AND USE THEM AS YE OUGHT, MAN:--
1)         BELIEVE ME, HAPPINESS IS SLY,
```

Again, the next matching line prints:

```
1)           AND COMES NOT AY WHEN SOUGHT, MAN.
```

The /B option to include blank lines in the comparison was not used in this example. Thus, SRCCOM recognizes only one more line before the end of file. Since the two identical lines do not constitute a match (three are needed) SRCCOM prints the last line as part of the difference section for the first file:

```
1)
1)                                    --SCOTTISH SONG
1)
****
```

In a similar manner, SRCCOM prints a differences section for the second file, ending the listing with the %FILES ARE DIFFERENT message.

NOTE

Regardless of the output specification, the differences message always prints on the terminal. If you compare two files that are identical and specify a file for the output listing, the message NO DIFFERENCES ENCOUNTERED prints on the terminal and SRCCOM does not create an output file.

PART V
ALTERING ASSEMBLED PROGRAMS

This part of the manual consists of the following three chapters: ODT, PATCH, and PAT. The three programs that these chapters describe can help you debug programs and make changes to programs that are already assembled.

Chapter 16 describes the on-line debugging technique (ODT). This program aids you in debugging assembly language programs. With ODT, you can control your program's execution, examine locations in memory and alter their contents, and search the object program for specific words.

Chapter 17 describes the PATCH utility program. PATCH can make code modifications to any RT-11 file. You use PATCH to examine and then change words or bytes in a file. PATCH's checksum feature is particularly useful when you are making a correction or improvement to an existing executable program; it verifies that the changes you make are correct.

Chapter 18 describes the object module patching utility (PAT). This program allows you to patch, or update, code in a relocatable binary object module. PAT accepts a file containing corrections or additional instructions and applies these corrections and additions to the original object module.

CHAPTER 16
ON-LINE DEBUGGING TECHNIQUE (ODT)

RT-11 on-line debugging technique (ODT) is a program (supplied with the system) that aids in debugging assembly language programs. From your terminal, you direct the execution of your program with ODT. ODT performs the following tasks:

- Prints the contents of any location for examination or alteration
- Runs all or any portion of an object program using the breakpoint feature
- Searches the object program for specific bit patterns
- Searches the object program for words that reference a specific word
- Calculates offsets for relative addresses
- Fills a single word, block of words, byte or block of bytes with a designated value.

Make sure you have an assembly listing and a link map available for the program you want to debug with ODT. You can make minor corrections to the program on line during the debugging session, and you can then execute the program under the control of ODT to verify the corrections. If you need to make major changes, such as adding a missing subroutine, note them on the assembly listing and incorporate them in a new assembly.

16.1 CALLING AND USING ODT

ODT is supplied as a relocatable object module. You can link ODT with your program (using the RT-11 linker) for an absolute area in memory and load it with your program. When you link ODT with your program, it is a good idea to link ODT low in memory relative to the program. If you link ODT high in memory, you must be sure that the buffer space for your program is contained within program bounds. Otherwise, if your program uses dynamic buffering, program execution can destroy ODT in memory. Figure 16-1 shows possible relationships between ODT and the program MYPROG in memory.

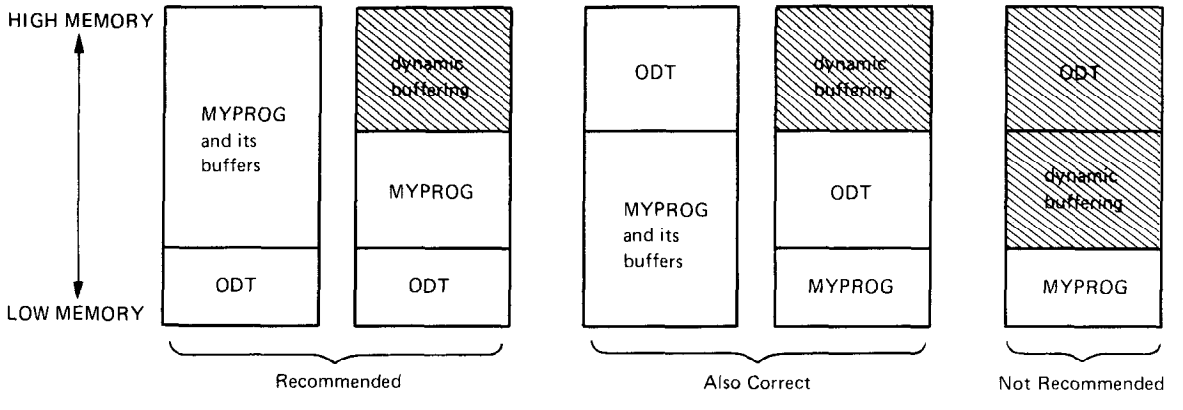


Figure 16-1 Linking ODT with a Program

Once loaded in memory with your program, ODT has three legal start or restart addresses. Use the lowest (O.ODT) for normal entry, retaining the current breakpoi :s. The next (O.ODT+2) is a restart address that clears all breakpoints and reinitializes ODT, thus saving the general registers and clearing the relocation registers. Use the last address (O.ODT+4) to reenter ODT. A reenter saves the processor status and general registers, and removes the breakpoint instructions from your program. ODT prints the bad entry (BE) error message. Breakpoints that were set are reset by the next ;G command. (;P is illegal after a BE message.) The ;G and ;P commands run a program; they are explained in Section 16.3.7.

The system uses as an absolute address the address of the entry point O.ODT shown in the linker load map.

NOTE

If you link ODT with an overlay-structured file, it should reside in the root segment so that it will always be in memory. Remove all breakpoints from the current overlay segment before execution proceeds to another overlay segment. A breakpoint inserted in an overlay is destroyed if it is overlaid during program execution.

The following examples show how to link and load ODT and how to restart ODT.

1. This example links ODT low in memory relative to MYPROG, creating the executable module MYPROG.SAV. Running MYPROG causes ODT to start automatically.

```
.LINK/MAP:TT:/DEBUG  MYPROG
RT-11 LINK  V03.01       Load Map          Mon 09-May-77 18:50:42
MYPROG.SAV       Title:  ODT      Ident:  X01.01

Section  Addr    Size     Global  Value    Global  Value    Global  Value

. ABS.   000000  001000    (RW,I,GBL,ABS,OVR)
         001000  016130    (RW,I,LCL,REL,CON)
                           O.ODT    001222

Transfer address = 001222, High limit = 017130 =  3884. words

.R MYPROG

 ODT  V01.04
*
```

2. This example links MYPROG low in memory relative to ODT and specifies O.ODT as the transfer address. Running MYPROG causes ODT to start automatically. The advantage to this method is that MYPROG is loaded at its normal execution-time address.

```
.LINK/MAP:TT: MYPROG,ODT/TRANSFER
Transfer address? O.ODT
RT-11 LINK  V03.01       Load Map          Mon 09-May-77 18:53:16
MYPROG.SAV       Title:  DEMOSP  Ident:  X01.01

Section  Addr    Size     Global  Value    Global  Value    Global  Value

. ABS.   000000  001000    (RW,I,GBL,ABS,OVR)
         001000  016130    (RW,I,LCL,REL,CON)
                           O.ODT    011260

Transfer address = 011260, High limit = 017130 =  3884. words
```

```
.R MYPROG

 ODT  V01.04
*
```

3. This example is similar to Example 2 above, except that execution does not automatically begin with ODT. When you start the program (MYPROG in this case) you must specify the address of O.ODT as shown in the link map.

```
.LINK/MAP:TT: MYPROG,ODT
RT-11 LINK  V03.01      Load Map        Mon 09-May-77 18:55:03
MYPROG.SAV      Title:  DEMOSP  Ident:  X01.01

Section  Addr    Size    Global  Value   Global  Value   Global  Value

. ABS.   000000  001000    (RW,I,GBL,ABS,OVR)
         001000  016130    (RW,I,LCL,REL,CON)
                           O.ODT   011260

Transfer address = 001036, High limit = 017130 =  3884. words

.GET MYPROG

.START 11260

 ODT  V01.04
*
```

4. This example links ODT with a bottom address of 4000, then loads ODT.SAV and MYPROG.SAV into memory. As in Example 3 above, when you start the program, you must specify the address of O.ODT as shown in the link map.

```
.LINK/MAP:TT: ODT/BOTTOM:4000
RT-11 LINK  V03.01      Load Map        Mon 09-May-77 18:59:42
ODT    .SAV     Title:  ODT     Ident:          /B:004000

Section  Addr    Size    Global  Value   Global  Value   Global  Value

. ABS.   000000  004000    (RW,I,GBL,ABS,OVR)
         004000  006072    (RW,I,LCL,REL,CON)
                           O.ODT   004222

Transfer address = 004222, High limit = 012072 =  2589. words

.GET ODT.SAV

.GET MYPROG.SAV

.START 4222

 ODT  V01.04
*
```

5. You can restart ODT by specifying O.ODT+2 as the start address. This reinitializes ODT and clears all breakpoints.

```
.START 4224

*
```

6. You can reenter ODT by specifying O.ODT+4 as the start address.

```
.START 4226

BE004242
*
```

If ODT is awaiting a command, a CTRL/C from the keyboard calls the RT-11 keyboard monitor. The monitor responds with ^C on the terminal and awaits a command. (You can use the monitor REENTER command to reenter ODT only if your program has set the reenter bit and ODT is linked high in memory relative to the program; otherwise, ODT is reentered at address O.ODT+4 as shown in Example 6 in Section 16.1.)

If you type CTRL/U during a search printout, the search terminates and ODT prints an asterisk.

16.2 RELOCATION

When the assembler produces a relocatable object module, the base address of the module is assumed to be location 000000. The addresses of all program locations, as shown in the assembly listing, are relative to this base address. After you link the module, many of the values and all of the addresses in the program are incremented by a constant whose value is the actual absolute base address of the module after it has been relocated. This constant is called the relocation bias for the module. Since a linked program can contain several relocated modules, each with its own relocation bias, and since, in the process of debugging, these biases have to be subtracted from absolute addresses continually in order to relate relocated code to assembly listings, RT-11 ODT provides automatic relocation.

The basis of automatic relocation is the eight relocation registers, numbered 0 through 7. You can set them to the values of the relocation biases at different times during debugging. Obtain relocation biases by consulting the link map. Once you set a relocation register, ODT uses it to relate relative addresses to absolute addresses. For more information on the exact nature of the relocation process, consult Chapter 11, Linker.

ODT evaluates a relocatable expression as a 16-bit (6-digit octal) number. You can type an expression in any one of the three forms presented in Table 16-1. In this table, the symbol n stands for an integer in the range 0 to 7 inclusive, and the symbol k stands for an octal number up to six digits long, with a maximum value of 177777. If you type more than six digits, ODT takes the last six digits typed, truncated to the low-order 16 bits. k can be preceded by a minus sign, in which case its value is the two's complement of the number typed. For example:

k (number typed)	Values
1	000001
-1	177777
400	000400
-177730	000050
1234567	034567

Section 16.3.13 describes the relocation register commands in greater detail.

Table 16-1 Forms of Relocatable Expressions (r)

Form	Expression	Value of r
A)	k	The value of k.
B)	n,k	The value of k plus the contents of relocation register n. (If the n part of this expression is greater than 7, ODT uses only the last octal digit of n.)
C)	C or C,k or n,C or C,C	Whenever you type the letter C, ODT replaces C with the contents of a special register called the constant register. (This value has the same role as the k or n that it replaces. The constant register is designated by the symbol $C and can be set to any value, as indicated below.)

16.3 COMMANDS AND FUNCTIONS

When ODT starts (as explained in Section 16.1) it indicates readiness to accept commands by printing an asterisk on the left margin of the terminal page. You can issue most of the ODT commands in response to the asterisk. You can examine a word and change it; you can run the object program in its entirety or in segments; you can search memory for specific words or references to them. The discussion below explains these features.

16.3.1 Printout Formats

Normally, when ODT prints addresses, it attempts to print them in relative form (Form B in Table 16-1). ODT looks for the relocation register whose value is closest to, but less than or equal to, the address to be printed. It then represents the address relative to the contents of the relocation register. However, if no relocation register fits the requirement, the address prints in absolute form. Since the relocation registers are initialized to -1 (the highest number), the addresses initially print in absolute form. If you change the contents of any relocation register, it can then, depending on the command, qualify for relative form.

For example, suppose relocation registers 1 and 2 contain 1000 and 1004 respectively, and all other relocation registers contain numbers much higher. In this case, the following sequence might occur (the slash command causes the contents of the location to be printed; the line feed command, LF, accesses the next sequential location):

```
*1000;1R                    sets relocation register 1 to 1000
*1,4;2R                     sets relocation register 2 to 1004
*774/000000 (LF)            opens location 774
000776 /007665 (LF)         opens location 776
1,000000 /000000 (LF)       opens absolute location 1000
1,000002 /000000 (LF)       opens absolute location 1002
2,000000 /000000            opens absolute location 1004
```

The printout format is controlled by the format register, $F. Normally, this register contains 0, in which case ODT prints addresses relatively whenever possible. You can open $F and change its contents to a non-zero value, however. In that case, all addresses print in absolute form (see Section 16.3.4, Accessing Internal Registers).

16.3.2 Opening, Changing, and Closing Locations

An open location is one whose contents ODT prints for examination, making those contents available for change. In a closed location, the contents are no longer available for change. Several commands are used for opening and closing locations.

Any command (except for the slash and backslash commands) that opens a location when another location is already open causes the currently open location to be closed. You can change the contents of an open location by typing the new contents followed by a single character command which requires no argument (i.e., LF, ^, RET, ←, @, >, <).

16.3.2.1 The Slash (/) – One way to open a location is to type its address followed by a slash. For example:

```
*1000/012746
```

This command opens location 1000 for examination and makes it ready to be changed.

If you do not want to change the contents of an open location, press the RETURN key to close the location. ODT prints an asterisk and waits for another command. However, to change the word, simply type the new contents before giving a command to close the location. For example:

```
*1000/012746 012345 (RET)
*
```

This command inserts the new value, 012345, in location 1000 and closes the location. ODT prints another asterisk indicating its readiness to accept another command.

Used alone, the slash reopens the last location opened. For example:

```
*1000/012345 2340 (RET)
*/002340
```

This command opens location 1000, changes its address to 002340, and then closes the location. ODT prints an asterisk, indicating its readiness to accept another command. The / character reopens the last location opened and verifies its value.

Remember that opening a location while another is open automatically closes the currently open location before opening the new location.

Also note that if you specify an odd-numbered address with a slash, ODT opens the location as a byte and subsequently behaves as if you had typed a backslash (see the following paragraph).

16.3.2.2 The Backslash (\) – In addition to operating on words, ODT operates on bytes. Typing the address of the byte followed by a backslash character opens the byte. (On the LT33 or LT35 terminal, type \ by pressing the SHIFT key while typing the L key.) This causes ODT to print the byte value at the specified address, to interpret the value as ASCII code, and to print the corresponding character, if possible, on the terminal. (ODT prints a ? when it cannot interpret the ASCII value as a printable character.)

```
*1001\101  =A
```

A backslash typed alone reopens the last open byte. If a word was previously open, the backslash reopens its even byte:

```
*1002/000004 \004  =?
```

16.3.2.3 The LINE FEED Key (LF) – If you type the LINE FEED key when a location is open, ODT closes the open location and opens the next sequential location:

```
*1000/002340(LF)
001002 /012740
```

In this example, the LINE FEED causes ODT to print the address of the next location along with its contents and to wait for further instructions. After the above operation, location 1000 is closed and 1002 is opened. You can modify the open location by typing the new contents.

If a byte location is open, typing a line feed opens the next byte location.

16.3.2.4 The Circumflex or Up-Arrow (^) – If you type the circumflex (or up-arrow) when a location is open (circumflex is produced on an LT33 or LT35 by typing SHIFT/N), ODT closes the open location and opens the previous location. To continue from the example above:

```
*001002/012740 ^
001000 /002340
```

This command closes location 1002 and opens location 1000. You can modify the open location by typing the new contents.

If the opened location is a byte, the circumflex opens the previous byte.

16.3.2.5 The Underline or Back-Arrow (←) – If you type the underline, or back-arrow, (use SHIFT/O on an LT33 or LT35 terminal) to an open word, ODT interprets the contents of the currently open word as an address indexed by the program counter (PC) and opens the addressed location:

```
*1006/000006 _
001016 /000405
```

Notice in this example that the open location, 1006, is indexed by the PC as if it were the operand of an instruction with addressing mode 67 (PC relative mode).

You can make a modification to the opened location before you type a line feed, circumflex, or underline. Also, the new contents of the location will be used for address calculations using the underline command. For example:

```
*100/000222 4(LF)      modifies to 4 and opens next location
000102 /000111 6^      modifies to 6 and opens previous location
000100 /000004 200_    changes to 200 and opens location indexed
000302 /123456         by PC
```

16.3.2.6 Open the Addressed Location (@) – You can use the at (@) symbol (SHIFT/P on the LT33 or LT35 terminal) to optionally modify a location, close it, and then use its contents as the address of the location to open next. For example:

```
*1006/001044 @          opens location 1044 next
001044 /000500

*1006/001044 2100@      modifies to 2100 and opens location
002100 /000167          2100
```

16.3.2.7 Relative Branch Offset (>) – The right-angle bracket, >, optionally modifies a location, closes it, and then uses its low-order byte as a relative branch offset to the next word to be opened. For example:

```
*1032/000407  301>      modifies to 301 and interprets as a
000636 /000010          relative branch
```

Note that 301 is a negative offset (-77). ODT doubles the offset before it adds it to the PC; therefore, 1034+(-176)=636.

16.3.2.8 Return to Previous Sequence (<) – The left-angle bracket, <, lets you optionally modify a location, close it, and then open the next location of the previous sequence that was interrupted by an underline, @, or right-angle bracket command. Note that underline, @, or right-angle bracket causes a sequence change to the open word. If a sequence change has not occurred, the left-angle bracket simply opens the next location as a LINE FEED does. This command operates on both words and bytes.

```
*1032/000407   301>          > causes a sequence change
000636 /000010 <             returns to original sequence
001034 /001040 @             @ causes a sequence change
001040 /000405 \005 = <      < now operates on byte
001035 \002 =↑ <             < acts like (LF)
001036 \004 =?
```

16.3.3 Accessing General Registers 0-7

Open the program's general registers 0-7 with a command in the following format:

```
$n/
```

The symbol n is an integer in the range 0-7 that represents the desired register. When you open these registers, you can examine them or change their contents by typing in new data as with any addressable location. For example:

```
*$0/000033 (RET)          examines register 0 then closes it
*

*$4/000474  464 (RET)     opens register 4, changes its contents
*                         to 000464, then closes the register
```

The example above can be verified by typing a slash in response to ODT's asterisk:

```
*/000464
```

You can use the LINE FEED, circumflex, underline or @ command when a register is open.

16.3.4 Accessing Internal Registers

The program's status register contains the condition codes of the most recent operational results and the interrupt priority level of the object program. Open it by typing $S. For example:

```
*$S/000311
```

$S represents the address of the status register. In response to $S in the example above, ODT prints the 16-bit word, of which only the low-order eight bits are meaningful. Bits 0-3 indicate whether a carry, overflow, zero, or negative (in that order) has resulted, and bits 5-7 indicate the interrupt priority level (in the range 0-7) of the object program. (Refer to the *PDP-11 Processor Handbook* for the status register format.)

You can also use the $ to open certain other internal locations listed in Table 16-2.

Table 16-2 Internal Registers

Register	Section	Function
$B	16.3.6	Location of the first word of the breakpoint table
$M	16.3.9	Mask location for specifying which bits are to be examined during a bit pattern search
$P	16.3.15	Location defining the operating priority of ODT
$S	16.3.4	Location containing the condition codes (bits 0-3) and interrupt priority level (bits 5-7)

(Continued on next page)

Table 16-2 (Cont.) Internal Registers

Register	Section	Function
$C	16.3.10	Location of the constant register
$R	16.3.13	Location of relocation register 0, the base of the relocation register table
$F	16.3.1	Location of the format register

16.3.5 Radix-50 Mode (X)

Many PDP-11 system programs employ the Radix-50 mode of packing certain ASCII characters three to a word. You can use Radix-50 mode by specifying the MACRO .RAD50 directive. ODT provides a method for examining and changing memory words packed in this way with the X command.

When you open a word and type the X command, ODT converts the contents of the opened word to its 3-character Radix-50 equivalent and prints these characters on the terminal. You can then type one of the responses from Table 16-3.

Table 16-3 Radix-50 Terminators

Response	Effect
RETURN key (RET)	Closes the currently open location
LINE FEED key (LF)	Closes the currently open location and opens the next one in sequence
Circumflex (^)	Closes the currently open location and opens the previous one in sequence
Any three characters whose octal code is 040 (space) or greater	Converts the three characters into packed Radix-50 format. Legal Radix-50 characters for this response are: . $ Space 0 through 9 A through Z

If you type any other characters, the resulting binary number is unspecified (that is, no error message prints and the result is unpredictable). You must type exactly three characters before ODT resumes its normal mode of operation. After you type the third character, the resulting binary number is available to be stored in the opened location. Do this by closing the location in any one of the ways listed in Table 16-3. For example:

```
*1000/042431  X=KBI  CBA(RET)
*1000/011421  X=CBA
```

NOTE

After ODT converts the three characters to binary, the binary number can be interpreted in one of many different ways, depending on the command that follows. For example:

```
*1234/063337  X=PRO  XIT/013704
```

Since the Radix-50 equivalent of XIT is 113574, the final slash in the example causes ODT to open location 113574 if it is a legal address.

16.3.6 Breakpoints

The breakpoint feature helps you monitor the progress of program execution. You can set a breakpoint at any instruction that is not referenced by the program for data. When a breakpoint is set, ODT replaces the contents of the breakpoint location with a BPT trap instruction so that program execution is suspended when a breakpoint is encountered. Then the original contents of the breakpoint location are restored, and ODT regains control.

With ODT, you can set up to eight breakpoints, numbered 0 through 7, at any one time. Set a breakpoint by typing the address of the desired location of the breakpoint followed by ;B. Thus, r;B sets the next available breakpoint at location r. (If all eight breakpoints have been set, ODT ignores the r;B command.) You can set or change specific breakpoints with the r;nB command, where n is the number of the breakpoint. For example:

```
*1020;B      sets breakpoint 0
*1030;B      sets breakpoint 1
*1040;B      sets breakpoint 2
*1032;1B     resets breakpoint 1
*
```

The ;B command removes all breakpoints. Use the ;nB command to remove only one of the breakpoints, where n is the number of the breakpoint. For example:

```
*;2B         removes the third breakpoint
*
```

ODT keeps a table of breakpoints; you can access that table. The $B/command opens the location containing the address of breakpoint 0. The next seven locations contain the addresses of the other breakpoints in order. You can sequentially open them by using the LINE FEED key. For example:

```
*$B/001020 LF
001136 /001032 LF
001140 /007070 LF
001142 /007070 LF
001144 /007070 LF
001146 /001046 LF
001150 /001066 LF
001152 /007070
```

In this example, breakpoint 0 is set to 1020, breakpoint 1 is set to 1032, breakpoint 5 is set to 1046, and breakpoint 6 is set to 1066. The other breakpoints are not set.

Note that a repeat count in a proceed command (;P) refers only to the breakpoint that ODT most recently encountered. Execution of other breakpoints is determined by their own repeat counts.

16.3.7 Running the Program (r;G and r;P)

ODT controls program execution. There are two commands for running the program: r;G and r;P. The r;G command starts execution (go) and r;P continues (proceed) execution after halting at a breakpoint. For example:

```
*1000;G
```

This command starts execution at location 1000. The program runs until it encounters a breakpoint or until it completes. If it gets caught in an infinite loop, it must be either restarted or reentered as explained in Section 16.1.

Upon execution of either the r;G or r;P command, the general registers 0-6 are set to the values in the locations specified as $0-$6. The processor status register is set to the value in the location specified as $S.

When ODT encounters a breakpoint, execution stops and ODT prints Bn; (where n is the breakpoint number), followed by the address of the breakpoint. You can then examine locations for expected data. For example:

```
*1010;3B        sets breakpoint 3 at location 1010
*1000;G         starts execution at location 1000
B3;001010       stops execution at location 1010
*
```

To continue program execution from the breakpoint, type ;P in response to ODT's last prompt (*).

When you set a breakpoint in a loop, you can allow the program to execute a certain number of times through the loop before ODT recognizes the breakpoint. Set a proceed count by using the k;P command. This command specifies the number of times the breakpoint is to be encountered before ODT suspends program execution (on the kth encounter). The count, k, refers only to the numbered breakpoint that most recently occurred. You can specify a different proceed count for the breakpoint when it is encountered. Thus:

```
B3;001010       halts execution at breakpoint 3
*1026;3B        resets breakpoint 3 at location 1026
*4;P            sets proceed count to 4 and
B3;001026       continues execution; the program loops
*               through the breakpoint three times and halts on
                the fourth occurrence of the breakpoint
```

Following the table of breakpoints (as explained in Section 16.3.6) is a table of proceed command repeat counts for each breakpoint. You can inspect these repeat counts by typing $B/ and nine line feeds. The repeat count for breakpoint 0 prints (the first seven line feeds causes the table of breakpoints to be printed; the eighth types the single instruction mode, explained in the next section, and the ninth line feed begins the table of proceed command repeat counts). The repeat counts for breakpoints 1 through 7 and the repeat count for the single-instruction trap follow in sequence. ODT initializes a proceed count to 0 before you assign it a value. After the command has been executed, it is set to -1. Opening any one of these repeat counts provides an alternative way of changing the count. Once the location is open, you can modify its contents in the usual manner by typing the new contents followed by the RETURN key. For example:

```
.
.
.
nnnnnn  /001036 (LF)        address of breakpoint 7
nnnnnn  /006630 (LF)        single instruction address
nnnnnn  /000000  15 (LF)    count for breakpoint 0; changes to 15
nnnnnn  /000000 (LF)        count for breakpoint 1
.
.
.
nnnnnn  /000000 (LF)        count for breakpoint 7
nnnnnn  /nnnnnn             repeat count for single instruction mode
```

Both the address indicated as the single instruction address and the repeat count for single instruction mode are explained in the following section.

16.3.8 Single Instruction Mode

With this mode, you specify the number of instructions to be executed before ODT suspends the program run. The proceed command, instead of specifying a repeat count for a breakpoint encounter, specifies the number of succeeding instructions to be executed. Note that breakpoints are disabled in single instruction mode. Table 16-4 lists the single instruction mode commands.

Table 16-4 Single Instruction Mode Commands

Command	Explanation
;nS	Enables single instruction mode. (n can be any digit and serves only to distinguish this form from the form ;S). Breakpoints are disabled.
n;P	Proceeds with program run for the next n instructions before reentering ODT. (If n is missing, it is assumed to be 1.) Trapping instructions and associated trap handlers can affect the proceed repeat count (see Section 16.4.2).
;S	Disables single instruction mode.

When the repeat count for single instruction mode is exhausted and the program suspends execution, ODT prints:

```
$B;nnnnnn
```

where nnnnnn is the address of the next instruction to be executed. The $B breakpoint table contains this address following that of breakpoint 7. However, unlike the table entries for breakpoints 0-7, direct modification has no effect.

Similarly, following the repeat count for breakpoint 7 is the repeat count for single instruction mode. You can modify this table entry directly. This is an alternative way of setting the single-instruction mode repeat count. In such a case, ;P implies the argument set in the $B repeat count table rather than an assumed 1.

16.3.9 Searches

With ODT, you can search all or any specific portion of memory for any bit pattern or for references to a particular location.

16.3.9.1 Word Search (r;W) – Before initiating a word search, you must specify the mask and search limits. The location represented by $M specifies the mask of the search. $M/ opens the mask register. The next two sequential locations (opened by LINE FEEDs) contain the lower and upper limits of the search. ODT examines in the search all bits set to 1 in the mask; it ignores other bits.

You must then give the search object and the initiating command, using the r;W command, where r is the search object. When ODT finds a match, (i.e., each bit set to 1 in the search object is set to 1 in the word ODT searches over the mask range) the matching word prints. For example:

```
*$M/000000 177400(LF)          tests high-order eight bits
nnnnnn  /000000  1000(LF)      sets low address limit
nnnnnn  /000000  1040(RET)     sets high address limit
*400;W                         initiates word search
001010 /000770
001034 /000404
*
```

In the above example, nnnnnn is an address internal to ODT; this location varies and is meaningful only for reference purposes. In the first line above, the slash was used to open $M, which now contains 177400; the LINE FEEDs open the next two sequential locations, which now contain the upper and lower limits of the search.

In the search process, ODT performs an exclusive OR (XOR) with the word currently being examined and the search object; the result is ANDed to the mask. If this result is 0, a match has been found and ODT reports it on the terminal. Note that if the mask is 0, all locations within the limits print. This provides a convenient method for dumping all memory locations within given limits using ODT.

Typing CTRL/U during a search printout terminates the search.

16.3.9.2 Effective Address Search (r;E) – ODT provides a search for words that reference a specific location. Open the mask register only to gain access to the low and high limit registers. After specifying the search limits (as explained for the word search), type the command r;E (where r is the effective address) to initiate the search.

Words that are an absolute address (argument r itself), a relative address offset, or a relative branch to the effective address print after their addresses. For example:

```
*$M/177400(LF)                    opens mask register only to gain
nnnnnn  /001000  1010(LF)         access to search limits
nnnnnn  /001040  1060(RET)
*1034;E                           initiates search
001016  /001006                   relative branch
001054  /002767                   relative branch
*1020;E                           initiates a new search
001022  /177774                   relative address offset
001030  /001020                   absolute address
```

Give particular attention to the reported effective address references. A word can have the specified bit pattern of an effective address without actually being used as one. ODT reports all possible references whether they are actually used or not.

Typing CTRL/U during a search printout terminates the search.

16.3.10 The Constant Register (r;C)

It is often desirable to convert a relocatable address into its value after relocation or to convert a number into its two's complement, and then to store the converted value in one or more places in a program. Use the constant register to perform this and other useful functions.

Typing r;C evaluates the relocatable expression to its 6-digit octal value, prints the value on the terminal, and stores it in the constant register. Invoke the contents of the constant register in subsequent relocatable expressions by typing the letter C. Examples follow:

```
*-4432;C=173346          places the two's complement of 4432 in the
                         constant register

*6632/062701  C(RET)     stores the contents of the constant register
                         in location 6632

*1000;1R                 sets relocation register 1 to 1000

*1,4272;C=005272         reprints relative location 4272 as an
                         absolute location and stores it in the
                         constant register
```

16.3.11 Memory Block Initialization (;F and ;I)

Use the constant register with the commands ;F and ;I to set a block of memory to a specific value. While the most common value required is 0, other possibilities are +1, -1, ASCII space, etc.

When you type the command ;F, ODT stores the contents of the constant register in successive memory words starting at the memory word address you specify in the lower search limit and ending with the address you specify in the upper search limit.

Typing the command ;I stores the low-order eight bits in the constant register in successive bytes of memory starting at the byte address you specify in the lower search limit and ending with the byte address you specify in the upper search limit.

For example, assume relocation register 1 contains 7000, 2 contains 10000, and 3 contains 15000. The following sequence sets word locations 7000-7776 to 0, and byte locations 10000-14777 to ASCII spaces:

```
*$M/000000 (LF)                    opens the mask register to gain
                                   access to search limits
nnnnnn  /000000  1,0 (LF)          sets the lower limit to 7000
nnnnnn  /000000  2,-2 (LF)         sets the upper limit to 7776
*0;C=000000                        sets the constant register to zero
*;F                                sets locations 7000-7776 to zero

*$M/000000 (LF)
nnnnnn  /007000  2,0 (LF)          sets the lower limit to 10000
nnnnnn  /007776  3,-1 (RET)        sets the upper limit to 14777
*40;C=000040                       sets the constant register to 40
                                   (space)
*;I                                sets the byte locations 10000-14777
*                                  to the value in the low-order 8
                                   bits of the constant register
```

16.3.12 Calculating Offsets (r;O)

Relative addressing and branching involve the use of an offset. An offset is the number of words or bytes forward or backward from the current location to the effective address. During the debugging session it is sometimes necessary to change a relative address or branch reference by replacing one instruction offset with another. ODT calculates the offsets in response to the r;O command.

The command r;O causes ODT to print the 16-bit and 8-bit offsets from the currently open location to address r. For example:

```
*346/000034  414;0 000044 022  22 (RET)
*/000022
```

This command opens location 346, calculates and prints the offsets from location 346 to location 414, changes the contents of location 346 to 22 (the 8-bit offset), and verifies the contents of location 346.

The 8-bit offset prints only if it is in the range -128(decimal) to 127(decimal) and the 16-bit offset is even, as was the case above. In the next example, the offset of a relative branch is calculated and modified so that it branches to itself.

```
*1034/103421  1034;0 177776 377  \021  377 (RET)
*/103777
```

Note that the modified low-order byte 377 must be combined with the unmodified high-order byte.

16.3.13 Relocation Register Commands

The use of the relocation registers is described briefly in Section 16.2. At the beginning of a debugging session, it is desirable to preset the registers to the relocation biases of those relocatable modules that will be receiving the most attention. Do this by typing the relocation bias, followed by a semicolon and the specification of relocation registers, as follows:

```
r;nR
```

The symbol r can be any relocatable expression, and n is an integer in the range 0-7. If you omit n, it is assumed to be 0. For example:

```
*1000;5R          puts 1000 into relocation register 5
*5,100;5R         adds 100 to the contents
*                 of relocation register 5
```

Once a relocation register is defined, you can use it to reference relocatable values. For example:

```
*2000;1R          puts 2000 into relocation register 1
*1,2176/002466    examines the contents of location 4176
*1,3712;0B        sets a breakpoint at location 5712
```

Sometimes programs can be relocated to an address below the one at which they were assembled. This could occur with PIC code (position independent code), which is moved without using the linker. In this case, the appropriate relocation bias would be the two's complement of the actual downward displacement. One method for easily evaluating the bias and putting it in the relocation register is illustrated in the following example.

Assume a program was assembled at location 5000 and was moved to location 1000. Then the following sequence enters the two's complement of 4000 in relocation register 1.

```
*1000;1R
*1,-5000;1R
*
```

Relocation registers are initialized to -1 so that unwanted relocation registers never enter into the selection process when ODT searches for the most appropriate register.

To set a relocation register to -1, type ;nR. To set all relocation registers to -1, type ;R.

ODT maintains a table of relocation registers, beginning at the address specified by $R. Opening $R ($R/) opens relocation register 0. Successively typing a LINE FEED opens the other relocation registers in sequence. When a relocation register is opened in this way, you can modify it as you would any other memory location.

16.3.14 The Relocation Calculators, nR and n!

When a location has been opened, it is often desirable to relate the relocated address and the contents of the location back to their relocatable values. To calculate the relocatable address of the opened location relative to a particular relocation bias, type:

```
n!
```

The symbol n specifies the relocation register. This calculator works with opened bytes and words. If you omit n, the relocation register whose contents are closest to, but less than or equal to, the opened location is selected automatically by ODT. In the following example, assume that these conditions are fulfilled by relocation register 3, which contains 2000. Use the following command to find the most likely module that a given opened byte is in:

```
*2500\011  =  !=3,000500
```

To calculate the difference between the contents of the opened location and a relocation register, type:

```
nR
```

The symbol n represents the relocation register. If you omit n, ODT selects the relocation register whose contents are closest to but less than or equal to the contents of the opened location. For example, assume the relocation bias stored in relocation register 1 is 7000:

```
*1,500/11032  1R=1,2032
```

The value 2032 is the content of 1,500, relative to the base 7000. The next example shows the use of both relocation calculators.

If relocation register 1 contains 1000, and relocation register 2 contains 2000, use the following command to calculate the relocatable addresses of location 3000 and its contents relative to 1000 and 2000:

```
*3000/006410  1!=1,002000  2!=2,001000  1R=1,5410  2R=2,4410
```

16.3.15 ODT Priority Level, $P

$P represents a location in ODT that contains the interrupt (or processor) priority level at which ODT operates. If $P contains the value 377, ODT operates at the priority level of the processor at the time ODT is entered. Otherwise, $P can contain a value between 0 and 7 corresponding to the fixed priority at which ODT operates.

To set ODT to the desired priority level, open $P. ODT prints the present contents, which you can then change:

```
*$P/000006  4 (RET)   lowers the priority to allow interrupts
*                     from the terminal
```

If you do not change $P, its value is seven.

You must set ODT's priority to 0 if you are using ODT in an FB environment while another job is running.

ODT does not always service breakpoints that are set in routines that run at different priority levels. For example, a program running at a low priority can use a device service routine that operates at a higher priority level. If you set $P low, ODT waits for terminal input at a low priority. If an interrupt occurs from a high priority routine, the breakpoints in the high priority routine are not recognized since they were removed when the earlier breakpoint occurred. That is, interrupts that are set at a priority higher than the one at which ODT is running are serviced, but any breakpoints are not recognized. To avoid this problem, set breakpoints at one priority level at a time. That is, set breakpoints within an interrupt service routine, but not at mainline code level. For a more complete discussion of how the PDP-11 handles priority and interrupts, refer to the processor handbook for your particular machine. ODT disables all breakpoints in the program whenever it gains control. Breakpoints are enabled when ;P and ;G commands are executed. For example:

```
*$P/00007  5
*1000;B
*2000;B
*1000;G
B0;001000
*                an interrupt occurs and is serviced
```

If a higher level interrupt occurs while ODT is waiting for input, the interrupt is serviced, and no breakpoints are recognized.

16.3.16 ASCII Input and Output (r;nA)

Inspect and change ASCII text by using a command of this syntax:

r;nA

The symbol r represents a relocatable expression, and n is a character count. If you omit n, it is assumed to be 1. ODT does not check the magnitude of n. ODT prints n characters starting at location r, followed by a carriage return/line feed combination. Table 16-5 lists responses and their effect.

Table 16-5 ASCII Terminators

Response	Effect
RETURN key (RET)	ODT outputs a carriage return/line feed combination followed by an asterisk and waits for another command.
LINE FEED key (LF)	ODT opens the byte following the last byte output.
Up to n characters of text	ODT inserts the text into memory, starting at location r. If you type fewer than n characters, terminate the command by typing CTRL/U. This causes a carriage return/line feed/asterisk combination to print. However, if you type exactly n characters, ODT responds with a carriage return/line feed combination, the address of the next available byte, and then a carriage return/line feed/asterisk combination.

16.4 PROGRAMMING CONSIDERATIONS

Information in this section is not necessary for the efficient use of ODT. However, it does provide a better understanding of how ODT performs some of its functions. In certain difficult debugging situations, this understanding is necessary.

16.4.1 Using ODT with Foreground/Background Jobs

It is possible to use ODT to debug programs written as either background or foreground jobs. ODT does not debug virtual tasks that use extended memory. In the background or under the single-job monitor, you can link ODT with the program as described in Example 1 in Section 16.1. To debug a program in the foreground area, DIGITAL recommends that you run ODT in the background while the program to be debugged is in the foreground. The sequence of commands to do this is as follows:

```
.FRUN PROG/P                loads the foreground program
LOADED AT nnnnnn            the first address of the job prints
.RUN ODT                    runs ODT in the background
ODT V01.01                  and sets a relocation register
*nnnnnn;0R                  to the start of the job

*$F/000000 0                clears the format register to enable
*0,nnnnnn;0B                proper address printing
                            sets a breakpoint

*0;G                        starts the keyboard monitor again

.RESUME                     starts the foreground job
```

The copy of ODT you use must be linked low enough so that it fits in memory along with the foreground job.

NOTE

Since ODT uses its own terminal handler, it cannot be used with the display hardware. If GT ON is in effect, ODT ignores it and directs its input and output only to the console terminal.

If you use ODT in a foreground/background environment while another job is running, set ODT's priority bit to 0 as follows:

```
**P/000007 0 (RET)
```

This puts ODT into the wait state at level 0, not at level 7. If you leave ODT's priority at 7, all interrupts (including clock) are locked out while ODT is waiting for terminal input.

16.4.2 Functional Organization

The internal organization of ODT is almost totally modularized into independent subroutines. The internal structure consists of three major functions: command decoding, command execution, and utility routines.

The command decoder interprets the individual commands, checks for command errors, saves input parameters for use in command execution, and sends control to the appropriate command execution routine.

The command execution routines take parameters saved by the command decoder and use the utility routines to execute the specified command. Command execution routines either return to the command decoder or transfer control to your program.

The utility routines are common routines such as SAVE-RESTORE and I/O. They are used by both the command decoder and the command executers.

16.4.3 Breakpoints

The function of a breakpoint is to give control to ODT whenever a program tries to execute the instruction at the selected address. Upon encountering a breakpoint, you can use all of the ODT commands to examine and modify the program.

When a breakpoint is executed, ODT removes all the breakpoint instructions from the code so that you can examine and alter the locations. ODT then types a message on the terminal in the form Bn;r, where r is the breakpoint address and n is the breakpoint number. ODT automatically restores the breakpoints when execution resumes.

There is a major restriction in the use of breakpoints: the program must not reference the word where a breakpoint was set since ODT altered the word. You should also avoid setting a breakpoint at the location of any instruction that clears the T-bit. For example:

```
MOV #240,177776      ;SET PRIORITY TO LEVEL 5
```

NOTE

Instructions that cause or return from traps (e.g., EMT, RTI) are likely to clear the T-bit, since a new word from the trap vector or the stack is loaded into the status register.

A breakpoint occurs when a trace trap instruction (placed in your program by ODT) is executed. When a breakpoint occurs, ODT operates according to the following algorithm:

1. Sets processor priority to 7 (automatically set by trap instruction).
2. Saves registers and sets up stack.

3. If internal T-bit trap flag is set, goes to step 13.
4. Removes breakpoints.
5. Resets processor priority to ODT's priority or user's priority.
6. Makes sure a breakpoint or single-instruction mode caused the interrupt.
7. If the breakpoint did not cause the interrupt, goes to step 15.
8. Decrements repeat count.
9. Goes to step 18 if non-zero; otherwise resets count to 1.
10. Saves terminal status.
11. Types message about the breakpoint or single-instruction mode interrupt.
12. Goes to command decoder.
13. Clears T-bit in stack and internal T-bit flag.
14. Jumps to the go processor.
15. Saves terminal status.
16. Types BE (bad entry) followed by the address.
17. Clears the T-bit, if set, in the user status and proceeds to the command decoder.
18. Goes to the proceed processor, bypassing the TT restore routine.

Note that steps 1-5 inclusive take approximately 100 microseconds. Interrupts are not permitted at this time, since ODT is running at priority level 7.

ODT processes a proceed (;P) command according to the following algorithm:

1. Checks the proceed for legality.
2. Sets the processor priority to 7.
3. Sets the T-bit flags (internal and user status).
4. Restores the user registers, status, and program counter.
5. Returns control to the user.
6. When the T-bit trap occurs, executes steps 1, 2, 3, 13, and 14 of the breakpoint sequence, restores breakpoints, and resumes normal program execution.

When a breakpoint is placed on an IOT, EMT, TRAP, or any instruction causing a trap, ODT follows this algorithm:

1. When the breakpoint occurs as described above, enters ODT.
2. When ;P is typed, sets the T-bit and executes the IOT, EMT, TRAP, or other trapping instruction.
3. Pushes the current PC and status (with the T-bit included) on the stack.
4. Obtains the new PC and status (no T-bit set) from the respective trap vector.
5. Executes the whole trap service routine without any breakpoints.
6. When an RTI is executed, restores the saved PC and PS (including the T-bit). Executes the instruction following the trap-causing instruction. If this instruction is not another trap-causing instruction, the T-bit trap occurs; reinserts the breakpoints in the user program, or decrements the single-instruction mode repeat count. If the following instruction is a trap-causing instruction, repeats this sequence starting at step 3.

NOTE

Exit from the trap handler must be by means of the RTI instruction. Otherwise, the T-bit is lost. ODT cannot regain control since the breakpoints have not yet been reinserted.

Note that the ;P command is illegal if a breakpoint has not occurred (ODT responds with ?). ;P is legal, however, after any trace trap entry.

The internal breakpoint status words have the following format:

1. The first eight words contain the breakpoint addresses for breakpoints 0-7. (The ninth word contains the address of the next instruction to be executed in single-instruction mode.)
2. The next eight words contain the respective repeat counts. (The following word contains the repeat count for single-instruction mode.)

You can change these words at will, either by using the breakpoint commands or by directly manipulating $B.

When program runaway occurs (that is, when the program is no longer under ODT control, perhaps executing an unexpected part of the program where you did not place a breakpoint) give control to ODT by pressing the HALT key to stop the computer and then restarting ODT (see Section 16.1). ODT prints an asterisk, indicating that it is ready to accept a command.

If the program you are debugging uses the console terminal for input or output, the program can interact with ODT to cause an error since ODT uses the console terminal as well. This interactive error does not occur when you run the program without ODT.

Note the following rules concerning the ODT break routine:

1. If the console terminal interrupt is enabled upon entry to the ODT break routine, and no output interrupt is pending when ODT is entered, ODT generates an unexpected interrupt when returning control to the program.
2. If the interrupt of the console terminal reader (the keyboard) is enabled upon entry to the ODT break routine, and the program is expecting to receive an interrupt to input a character, both the expected interrupt and the character are lost.
3. If the console terminal reader (keyboard) has just read a character into the reader data buffer when the ODT break routine is entered, the expected character in the reader data buffer is lost.

16.4.4 Searches

The word search lets you search for bit patterns in specified sections of memory. Using the $M/ command, specify a mask, a lower search limit ($M+2), and an upper search limit ($M+4). Specify the search object in the search command itself.

The word search compares selected bits (where 1s appear in the mask) in the word and search object. If all of the selected bits are equal, the unmasked word prints.

The search algorithm is as follows:

1. Fetches a word at the current address.
2. XORs (exclusive OR) the word and search object.
3. ANDs the result of step 2 with the mask.
4. If the result of step 3 is 0, types the address of the unmasked word and its contents; otherwise, proceeds to step 5.
5. Adds 2 to the current address. If the current address is greater than the upper limit, types * and returns to the command decoder; otherwise, goes to step 1.

Note that if the mask is 0, ODT prints every word between the limits, since a match occurs every time (i.e., the result of step 3 is always 0).

In the effective address search, ODT interprets every word in the search range as an instruction that is interrogated for a possible direct relationship to the search object. The mask register is opened only to gain access to the search limit registers.

The algorithm for the effective address search is as follows ((X) denotes contents of X, and K denotes the search object):

1. Fetches a word at the current address X.
2. If (X)=K [direct reference], prints contents and goes to step 5.
3. If (X)+X+2=K [indexed by PC], prints contents and goes to step 5.
4. If (X) is a relative branch to K, prints contents.
5. Adds 2 to the current address. If the current address is greater than the upper limit, performs a carriage return/line feed combination and returns to the command decoder; otherwise, goes to step 1.

16.4.5 Terminal Interrupt

Upon entering the TT SAVE routine, ODT follows these steps:

1. Saves the LSR status register (TKS).
2. Clears interrupt enable and maintenance bits in the TKS.
3. Saves the TT status register (TPS).
4. Clears interrupt enable and maintenance bits in the TPS.

To restore the TT:

1. Wait for completion of any I/O from ODT.
2. Restore the TKS.
3. Restore the TPS.

NOTES

1. If the TT printer interrupt is enabled upon entry to the ODT break routine, the following can occur:

   a. If no output interrupt is pending when ODT is entered, an additional interrupt always occurs when ODT returns control to the user.
   b. If an output interrupt is pending upon entry, the expected interrupt occurs when the user regains control.

2. If the TT reader (keyboard) is busy or done, the expected character in the reader data buffer is lost.

3. If the TT reader (keyboard) interrupt is enabled upon entry to the ODT break routine, and a character is pending, the interrupt (as well as the character) is lost.

16.5 ERROR DETECTION

ODT detects two types of error: illegal or unrecognizable command and bad breakpoint entry. ODT does not check for the legality of an address when you command it to open a location for examination or modification. Thus the command:

```
177774/
?MON-F-Trap to 4 003362
```

references nonexistent memory, thereby causing a trap through the vector at location 4. If the program you are debugging with ODT has requested traps through location 4 with the .TRPSET EMT, the program receives control at its TRPSET address.

Typing something other than a legal command causes ODT to ignore the command and to print:

```
(echoes illegal command) ?
*
```

and to wait for another command. Therefore, to cause ODT to ignore a command just typed, type any illegal character (such as 9 or RUBOUT) and the command will be treated as an error and ignored.

ODT suspends program execution whenever it encounters a breakpoint (that is, traps to its breakpoint routine). If the breakpoint routine is entered and no known breakpoint caused the entry, ODT prints:

```
BEnnnnnn
*
```

and waits for another command. BEnnnnnn denotes bad entry from location nnnnnn. A bad entry may be caused by an illegal trace trap instruction, by a T-bit set in the status register, or by a jump to some random location within ODT.

CHAPTER 17
PATCH

You can use the PATCH utility program to make code modifications to any RT-11 file (see Table 3-2 in this manual for a complete list of RT-11 file types). You use PATCH to interrogate and then to change words or bytes in the file.

It is always a good idea to create a backup version of the file you want to patch, because PATCH makes changes directly to the file as you work.

17.1 CALLING AND USING PATCH

To call PATCH from the system device, respond to the dot (.) printed by the keyboard monitor by typing:

```
R PATCH (RET)
```

PATCH then prints:

```
FILE NAME --
*
```

You should enter the name of the file you want to patch according to this general syntax:

```
filespec[/option . . .]
```

where:

filespec — represents the device, file name, and file type of the file you want to patch.

/option — is one of the options listed in Table 17-1.

If you do not specify a file type, PATCH assumes a .SAV file type.

17.1.1 PATCH Options

Table 17-1 summarizes the options that are valid for PATCH at this point in the opening command.

Table 17-1 PATCH Options

Option	Meaning
/A	Use with a device specification with or without a file specification. Use without a file specification to repair damaged RT-11 directories on directory-structured devices or to patch the bootstrap on disk block 0. Use with a file specification when the file is a source file or has a file type other than .SAV.
/M	Use if the file is an RT-11 monitor file.
/O	Use if the file is an overlay-structured file.
/C	Requires you to enter a checksum. If you make no modifications, PATCH ignores the /C option.
/D	Use if you do not know the checksum for a particular patch. PATCH prints the checksum for that patch. If you make no modifications, PATCH ignores the /D option.

Note that you must enter the complete file specification and accompanying options at this point; they are not legal at any other time. If you enter a carriage return instead of a file specification, however, PATCH prints its current running version number. It then repeats the prompt for a file specification.

After you enter the file specification, PATCH prints another asterisk and waits for commands.

17.1.2 Checksum

The checksum option helps you verify your work. It lets you compare the patch that you make to another patch that is known to be correct. The checksum does not tell you specifically where your error is, but it does tell you that an inconsistency exists.

PATCH can maintain a running total of the value of each command, argument, and character you enter. This total is called the checksum for the patch.

For example, if you receive from DIGITAL a patch to improve your system's performance, the patch contains a checksum value. You should use the /C option in the first PATCH command line, then make the modifications to your file exactly as shown in the DIGITAL patch. When you exit, PATCH asks you for a checksum. Enter the value from the DIGITAL patch. If the checksum you enter and the checksum that PATCH generated when you made your modifications do not match, PATCH prints the ?PATCH-W-CHECKSUM ERROR message. You then know that you made an error in patching your file, and that you need to try again.

17.2 PATCH COMMANDS

Table 17-2 summarizes the PATCH commands. Upper case characters represent PATCH commands; lower case characters represent octal values or ASCII characters. The following sections describe the commands in detail. Section 17.3 provides examples that use PATCH.

17.2.1 Patching a New File (F)

The F command causes PATCH to request you to enter a checksum, or it prints the required checksum (depending upon the options you specify). It also causes PATCH to close the currently open file and to print an asterisk indicating its readiness to accept another command string. No checksum dialogue is invoked if you have not previously specified checksum options (with /D or /C).

17.2.2 Exiting from Patch (E)

The E command causes PATCH to close the currently open file after printing the checksum dialogue according to the options you specify and return control to the RT-11 monitor. As with the F command, the checksum dialogue is by-passed if you have not specified checksum options.

17.2.3 Examining and Changing Locations in the File

For a non-overlay file, you can open a word address (as with ODT) by typing:

[relocation register,] offset/

PATCH types the contents of the location and waits for you to enter either a new location contents or another command.

For an overlay file, the format is:

[segment number:] [relocation register,] offset/

Segment number represents the overlay segment number as it is printed on the link map for the file. If you omit the segment number, PATCH assumes the root segment. If you make an error in a command string while patching an overlaid program, you can use CTRL/U to cancel the command. However, PATCH assumes the entire line is incorrect and preserves only the previously set relocation registers. PATCH preserves the segment number only across the ^ and (LF) commands.

Table 17-2 PATCH Commands

Command	Section	Explanation
v;nR	17.2.8	Sets relocation register n to value v.
x;B	17.2.7	Sets the bottom address of the overlay file to the value x.
r,o/	17.2.3	Opens the word location indicated by the contents of relocation register r plus offset o.
r,o\	17.2.3	Opens the byte location indicated by the contents of relocation register r plus offset o.
s:r,o/	17.2.3	Opens the word location indicated by the contents of relocation register r plus offset o in overlay segment s.
s:r,o\	17.2.3	Opens the byte location indicated by the contents of relocation register r plus offset o in overlay segment s.
(RET)	17.2.3	Closes the currently open word or byte.
(LF)	17.2.3	Closes the currently open word or byte and opens the next sequential word or byte.
n	17.2.3	Closes the currently open word or byte and opens the previous word or byte.
@	17.2.3	Closes the currently open word and opens the word it addresses.
F	17.2.1	Closes the file currently open and requests a new file specification.
E	17.2.2	Closes the file currently open and returns control to RT-11 monitor.
x;o	17.2.5	Indicates that a value in the overlay handler or its tables is being modified to the value x and that the overlay structure must be reinitialized. A value of 0 is illegal and generates an error message.
&	17.2.6	Indicates that PATCH should add the contents of all subsequently opened locations to the checksum, until it encounters another & symbol.
A	17.2.4	Prints the contents of the opened word or byte as ASCII characters (if a byte is open, one character prints; if a word is open, two characters print).
X	17.2.4	Prints the contents of the opened word as an unpacked Radix-50 word.
C(x[x])	17.2.4	Resets the contents of the opened word or byte to the ASCII value you type (if a byte is open, you must type one character; if a word is open, you must type two characters).
P(xxx)	17.2.4	Resets the contents of the currently opened word to the packed Radix-50 value of the three ASCII characters you type (you must type three characters).

Similarly, you can open a byte address in a file. The format for non-overlay files is:

[relocation register,] offset\

The format for overlay files is:

[segment number:] [relocation register,] offset\

Once a location has been opened, you can optionally type in the new contents in the format:

[relocation register,] octal value

Follow this line by one of the control characters from Table 17-3.

Table 17-3 PATCH Control Characters

Character	Function
(RET)	Closes the current location by changing contents to the new contents (if any) and awaits additional control input.
(LF)	Closes the current location by changing its content to the new contents (if any) and opens the next sequential word or byte.
^	Closes the current location by changing its contents to the new contents (if any) and opens the previous word or byte.
@	Closes the current word location and opens the word it addresses (in the same segment if it is an overlay file).

17.2.4 Translating and Indirectly Modifying Locations with a File

After opening a location within a file, you can translate the contents into ASCII characters or into the equivalent of a Radix-50 packed word.

To obtain the ASCII equivalent of the opened location, type the following command after PATCH prints the contents in octal.

A

PATCH then translates the word or byte into two (or one, if a byte is opened) ASCII characters. In this example, a byte is opened:

```
*1,100\ 102      A = B(LF)
```

PATCH prints only the printable ASCII characters in the opened word or byte (all non-printing characters, such as ASCII codes 0-37, are represented by the ? character). In this example, a word is opened:

```
*1, 100/ 302     A = B?(LF)
```

In the next example, a word is opened, and both ASCII characters are printable:

```
*1, 100/ 33502     A = B7(RET)
```

In these examples, one or both of the characters cannot be printed:

```
*0, 400/ 466      A = 6? (LF)

*1, 202/ 55001    A = ?Z (LF)

*616/ 401         A = ?? (RET)
```

To unpack a Radix-50 word as three ASCII characters, type the following command after PATCH prints the contents of the opened word.

```
X
```

PATCH then unpacks the opened word and prints three ASCII characters.

Note that you must open a word and not a byte.

If the word you open contains an illegal Radix-50 word, PATCH prints ???. If the translated character is not printable, PATCH prints ? in place of it.

Neither the A command nor the X command alters the contents of the open location; however, PATCH updates the checksum to reflect the fact that you have entered a new command.

You can specify the A and X commands in any order on the same command line without altering the contents of the open location. For example,

```
*1, 15022/     50553   X = MAC  A = kQ
```

After examining the location with the A or X command, you can change the location if you wish. For example,

```
*45660/10146   A = F?   X = BX8  12122 (RET) or (LF)
```

If the same location is reopened, the following change appears:

```
*45660/12122
```

You can change the contents of a location to the ASCII code of the value you specify by using the C command. You can use the P command to change a word to the packed Radix-50 word of the three characters you specify. This example changes an open byte to the ASCII code for the letter Z:

```
*1, 115\ 101      C (Z) (RET)
```

Note that PATCH prints the parentheses itself; you type only the character Z.

When reopened, that byte contains the ASCII code for Z:

```
*1, 115\ 132
```

Similarly, PATCH inserts the ASCII code for two ASCII characters into the low order and high order bytes, respectively, of one word. This example changes an open word to the ASCII code for AZ:

```
*0,10116/ 103523   C (AZ) ^
```

If reopened, the location contains the ASCII code for AZ:

```
*0,10116/ 55101    A = AZ
```

You can examine the same location in more conventional ways, as this example shows:

```
*0,10116\ 101(LF)
0,10117\ 132
```

Similarly, you can use the P command to change the contents of an open word to the Radix-50 packed word equivalent of the three ASCII characters you specify. This example changes the Radix-50 word equivalent of SAV to REL:

```
*21,400/ 73376   x = SAV   P (REL)(RET)
```

17.2.5 Setting Values in the Overlay Handler Tables of a Program

Use the ;O command to effect any changes to the overlay handler tables in an overlaid program. For example,

```
*616/    1043      1100;0
*
```

This command line increases the size of the referenced overlay region by 35(8) words or 58(10) bytes, to allow room for a patch. The value being modified is a value associated with the overlay handler tables, or a value required by the overlay handler for proper overlay structure initialization. The overlay structure is reinitialized and you can enter commands to modify the new region on the next line. A value of 0 is not permitted with the ;O command. If you omit the preceding argument, or use 0, an error message prints on the terminal.

17.2.6 Including the Old Contents Into the Checksum

Sometimes it is important that the present contents of the locations being changed have known specific values. This is the case when DIGITAL publishes system patches. The & command is designed to aid in implementing system patches. It automatically includes the old contents of an open location into the checksum. This command is a simple switch. The first occurrence of the & turns the switch on, the second turns it off. While the switch is on, the old contents of every location you open and close properly become part of the checksum. To use the & command, type:

```
&
```

Patch then prints a carriage return-line feed sequence and another * indicating its readiness to accept another command. This switch is then enabled.

If you type the command on a line where a location is currently open, PATCH closes the location and resets the switch. PATCH then prompts with an asterisk indicating that it is ready to accept additional commands.

17.2.7 Setting the Bottom Address

To patch an overlay file, PATCH must know the bottom address at which the program was linked, if it is different from the initial stack pointer. This is the case if the program sets location 42 in an .ASECT. To set the bottom address, type:

```
bottom address;B
```

You must issue the B command before you open any locations in an overlay for modification.

17.2.8 Setting Relocation Registers

You set the relocation registers 0-7 (as with ODT) with the R command. The R command has the syntax:

relocation value;relocation registerR

Be careful when you type this command string. If you inadvertently substitute a comma (,) for the semicolon (;) in the R command, PATCH does not generate an error message. However, it does not set the value you specify in the relocation register.

Once you set one of the eight relocation registers, the expression:

relocation register,octal number

in a command string will have the value:

relocation value + octal number

17.3 PATCH EXAMPLES

This section consists of two patch examples: one example for a non-overlaid file, and one example for an overlaid file. In each case, the steps that are taken to assemble, link, and patch the files are clearly illustrated.

The following command assembles the MACRO program PROMPT.MAC:

```
.MACRO/LIST PROMPT
ERRORS DETECTED:  0
```

The following listing is produced on the line printer as a result of the assembly. It consists of two parts: 1) the assembly listing of the source program and 2) the symbol table listing.

```
PROMPT.MAC      MACRO V03.00 5-MAY-77 16:54:30 PAGE 1

      1                                                      .TITLE  PROMPT.MAC
      2                                                      .MCALL  .PRINT,.EXIT
      3                                                      .MCALL  .TTYOUT,.TTYIN
      4 000000                                               .CSECT  HGHSEG
      5                                                      .NLIST  BEX
      6 000000  112700  000052          START:  MOVB    #'*,R0          ; PRINT,...
      7 000004                                 .TTYOUT                 ; ...A PROMPT.
      8 000010                                 .TTYIN                  ; ACCEPT A CHARACTER FROM THE KEYBOARD
      9 000014  122700  000040                 CMPB    #' ,R0          ; IS IT A CONTROL CHARACTER?
     10 000020  101367                         BHI     START           ; YES - MUST BE A MISTAKE.
     11 000022  122700  000057                 CMPB    #'/,R0          ; NO - IS IT A "/"?
     12 000026  001011                         BNE     ERROR           ; NO - REPORT THE ERROR.
     13 000030                                 .TTYIN                  ; YES - GET NEXT.
     14 000034  122700  000126                 CMPB    #'V,R0          ; IS IT A "V" CHARACTER?
     15 000040  001004                         BNE     ERROR           ; NO - REPORT THE ERROR.
     16 000042                                 .PRINT  #MSG            ; YES - PRINT THE VERSION MESSAGE...
     17 000050                          EXIT:  .EXIT                   ; ...AND THEN EXIT TO THE RT-11 MONITOR.
     18 000052                          ERROR: .PRINT  #CMDERR          ; PRINT THE "?(RET)(LF)",...
     19 000060  000747                         BR      START           ; ...AND THEN RESTART.
     20 000062     077     000          CMDERR: .ASCIZ  /?/
     21 000064     016     012     106  MSG:    .ASCIZ  <16><12>/FILE  V03.01
     22                                                      .LIST   BEX
     23          000000'                                     .END    START


PROMPT.MAC      MACRO V03.00 5-MAY-77 16:54:30 PAGE 1-1
SYMBOL TABLE

CMDERR  000062R     002 ERROR   000052R     002 EXIT    000050R     002 MSG     000064R     002 START  000000R     002

. ABS.  000000      000
        000000      001
HGHSEG  000107      002
ERRORS DETECTED:  0

VIRTUAL MEMORY USED:  562 WORDS ( 3 PAGES)
DYNAMIC MEMORY AVAILABLE FOR  58 PAGES
DK:PROMPT,LP:PROMPT=DK:PROMPT.MAC/C
```

The next command links file PROMPT.OBJ and produces an executable module called PROMPT.SAV.

```
.LINK/MAP  PROMPT
```

The following listing is produced on the line printer as a result of the link operation.

```
RT-11 LINK  V03.01      LOAD MAP        THU 05-MAY-77 16:55:28
PROMPT.SAV      TITLE:  PROMPT  IDENT:

SECTION  ADDR   SIZE    GLOBAL  VALUE   GLOBAL  VALUE   GLOBAL  VALUE

. ABS.  000000  001000   (RW,I,GBL,ABS,OVR)
HGHSEG  001000  000110   (RW,I,GBL,REL,OVR)

TRANSFER ADDRESS = 001000, HIGH LIMIT = 001110 =  292. WORDS
```

The program PROMPT has an error. On line 21 the characters <16> should really be <15> . The following example uses PATCH to correct the error.

```
.R PATCH

FILE NAME--
*PROMPT/D
*1000;1R
*1,64\  16      15
*E

?PATCH-I-Checksum = 30633
.
```

The example shown above uses the /D option, which requests PATCH to print the checksum when the operation completes. Next, relocation register 1 is set to the transfer address, which the link map shows is 1000. The next command opens relative location 64, which contains the error, as the assembly listing shows at line sequence number 21. The value 15 is substituted for 16 (by typing 15 followed by a carriage return) and the exit command is issued (with E). PATCH then prints the checksum for the operation. It is 30633.

The next example verifies the change just made.

```
.R PATCH

FILE NAME--
*PROMPT
*1000;1R
*1,64\  15
1,65\   12
1,66\   106
1,67\   111
1,70\   114
1,71\   105
1,72\   40
1,73\   40
1,74\   126
1,75\   60
1,76\   63
```

```
1,77\   56
1,100\  60
1,101\  61
1,102\  40
1,103\  40
1,104\  15
1,105\  12
*E
.
```

As before, relocation register 1 is set to 1000 and location 64 is opened. Now it contains the correct value, 15. The rest of the command lines are terminated with a line feed. This closes the open location and opens the next one. This example shows the values from line 21 of the assembly listing (RET) (LF) FILE V03.01 (RET) (LF) as they are stored in memory.

The following commands assemble two MACRO programs: PTCH.MAC, the main program, and OVRLAY.MAC, the overlay.

```
.MACRO/LIST PTCH
ERRORS DETECTED:  0

.MACRO/LIST OVRLAY
ERRORS DETECTED:  0
.
```

The following listings were produced by the two assemblies.

```
PTCH.MAC        MACRO V03.00 5-MAY-77 17:58:35 PAGE 1

      1                                                     .TITLE  PTCH.MAC
      2                                                     .MCALL  .PRINT,.EXIT
      3 000000                                              .CSECT  HGHSEG
      4                                                     .GLOBL  ENTRY,MSG1
      5 000000  000403                  START:  BR      EXIT            ; BRANCH IMMEDIATELY TO CALL OVERLAY.
      6 000002  012700  000016'                 MOV     #MSG,R0         ; ALTERNATIVELY PRINT A MESSAGE
      7 000006                                  .PRINT                  ; DO THE PRINT.
      8 000010  004767  000000G         EXIT:   JSR     PC,ENTRY        ; CALL IN THE OVERLAY.
      9 000014                                  .EXIT                   ; THEN EXIT ON RETURN.
     10                                                     .NLIST  BEX
     11 000016      015     012     124  MSG:    .ASCIZ  <15><12>/THIS IS A SUCCESSFUL PATCH/<15><12>
     12 000055      015     012     124  MSG1:   .ASCIZ  <15><12>/THIS IS AN OVERLAY PATCH/<15><12>
     13                                                     .LIST   BEX
     14          000000'                                   .END    START


PTCH.MAC        MACRO V03.00 5-MAY-77 17:58:35 PAGE 1-1
SYMBOL TABLE

ENTRY = ****** G        EXIT    000010R     002 MSG     000016R    002 MSG1    000055RG    002 START   000000R    002

. ABS.  000000      000
        000000      001
HGHSEG  000112      002
ERRORS DETECTED:  0

VIRTUAL MEMORY USED:  509 WORDS  ( 2 PAGES)
DYNAMIC MEMORY AVAILABLE FOR  59 PAGES
DK:PTCH,LP:PTCH=DK:PTCH


OVRLAY.MAC      MACRO V03.00 5-MAY-77 17:58:57 PAGE 1

      1                                                     .TITLE  OVRLAY.MAC
      2                                                     .MCALL  .PRINT
      3 000000                                              .CSECT  OVLSEG
      4                                                     .GLOBL  MSG1,ENTRY
      5 000000  000403                  ENTRY:  BR      RETURN          ; BRANCH IMMEDIATELY TO RETURN.
      6 000002  012700  000000G                 MOV     #MSG1,R0        ; ALTERNATIVELY PRINT A MESSAGE
      7 000006                                  .PRINT
      8 000010  000207                  RETURN: RTS     PC              ; THEN RETURN.
      9         000001                                          .END
```

```
OVRLAY.MAC      MACRO V03.00 5-MAY-77 17:58:57 PAGE 1-1
SYMBOL TABLE

ENTRY   000000RG    002 MSG1  = ****** G        RETURN  000010R    002

. ABS.  000000      000
        000000      001
OVLSEG  000012      002
ERRORS DETECTED:  0

VIRTUAL MEMORY USED:  354 WORDS  ( 2 PAGES)
DYNAMIC MEMORY AVAILABLE FOR  59 PAGES
DK:OVRLAY,LP:OVRLAY=DK:OVRLAY
```

The next command links the module PTCH.OBJ and the overlay module OVRLAY.OBJ, producing the executable module ROOT.SAV.

```
.LINK/MAP/PROMPT/EXECUTE:ROOT PTCH
*OVRLAY/O:1
*//

.
```

The following listing is the load map that results from this link.

```
RT-11 LINK  V03.01      LOAD MAP        THU 05-MAY-77 17:59:51
ROOT  .SAV      TITLE:  PTCH.M  IDENT:

SECTION  ADDR   SIZE    GLOBAL  VALUE   GLOBAL  VALUE   GLOBAL  VALUE

. ABS.  000000  001122   (RW,I,GBL,ABS,OVR)
HGHSEG  001122  000112   (RW,I,GBL,REL,OVR)
                       MSG1    001177
 SEGMENT SIZE = 001234 =   334. WORDS
 OVERLAY REGION 000001  SEGMENT 000001
OVLSEG  001236  000012   (RW,I,GBL,REL,OVR)
                       ENTRY @ 001236
 SEGMENT SIZE = 000012 =     5. WORDS


TRANSFER ADDRESS = 001122, HIGH LIMIT = 001250 =  340. WORDS
```

The following example shows how to patch an overlay segment.

```
.R PATCH

FILE NAME--
*ROOT/O/C
*1236;1R
*1:1,0/ 403      240
*E

Checksum?   45475

.R ROOT

THIS IS AN OVERLAY PATCH


.
```

The options /O and /C are used in the file specification. /O indicates that the file is overlaid. /C causes PATCH to verify that the changes are correct by requesting a checksum value, which it compares to the actual checksum value the changes generate internally. The patch for this example was supplied by an experienced user, and the checksum for the correct patch is known to be 45475.

The first command line sets relocation register 1 to the start of the overlay segment to be patched. The load maps show that overlay segment 1 begins at location 1236. The next command opens the first location in overlay segment 1. It contains a branch instruction (403). A no-op instruction is substituted for it (240) followed by a carriage return, and E is used to exit. PATCH then requests the checksum value and 45475 is entered. This matches the checksum that the changes generated internally, so control returns to the monitor and the patch is successful.

The program is executed by typing:

R ROOT (RET)

Control branches immediately to the overlay segment. Since the branch instruction at ENTRY: is now inoperative, control passes to the next line and the message prints on the terminal.

NOTE

The linker allocates space for overlay segments in 256-word blocks. Each segment begins on a block boundary. If a particular overlay segment's size is less than a whole number multiple of 256, you can add patches in the free space that exists between the end of that overlay segment and the beginning of the next block. To do this you must first modify the word-count word in the overlay handler table so that the patches you add are included in the size of the overlay segment. Be careful not to patch into the next block, though, because the next overlay segment begins there.

CHAPTER 18
OBJECT MODULE PATCH UTILITY (PAT)

PAT, the RT-11 object module patch utility, allows you to patch, or update, code in a relocatable binary object module. PAT does not permit you to examine the octal contents of an object module; use PATCH (described in Chapter 17) to do that. PAT makes the patch to the object module by means of the procedure outlined in Figure 18-2. One advantage to using PAT is that you can add relatively large patches to an object module without performing any octal calculations. PAT accepts a file containing corrections or additional instructions and applies these corrections and additions to the original object module. You prepare the correction input in MACRO source form and assemble it with the MACRO-11 assembler.

Input to PAT is two files: 1) the original input file and 2) a correction file containing the corrections and additions to the input file. The input file consists of one or more concatenated object modules. You can correct only one of these object modules with a single execution of the PAT utility. The correction file consists of object code that, when linked by the linker, either replaces or appends to the original object module. Output from PAT is the updated input file.

18.1 CALLING AND USING PAT

To call PAT from the system device, respond to the dot (.) printed by the keyboard monitor by typing:

```
R PAT (RET)
```

The Command String Interpreter prints an asterisk at the left margin on the console terminal when it is ready to accept a command line. Chapter 6 describes the general syntax of the command line that PAT accepts.

Type two CTRL/Cs to halt PAT at any time (or a single CTRL/C to halt PAT when it is waiting for console terminal input) and return control to the monitor. To restart PAT, type R PAT in response to the monitor's dot. When PAT executes an operation it returns control to the RT-11 monitor.

Figure 18-1 shows how you use PAT to update a file (FILE1) consisting of three object modules (MOD1, MOD2, and MOD3) by appending a correction file to MOD2. After running PAT, you use the linker to relink the updated module with the rest of the file and to produce a corrected executable program.

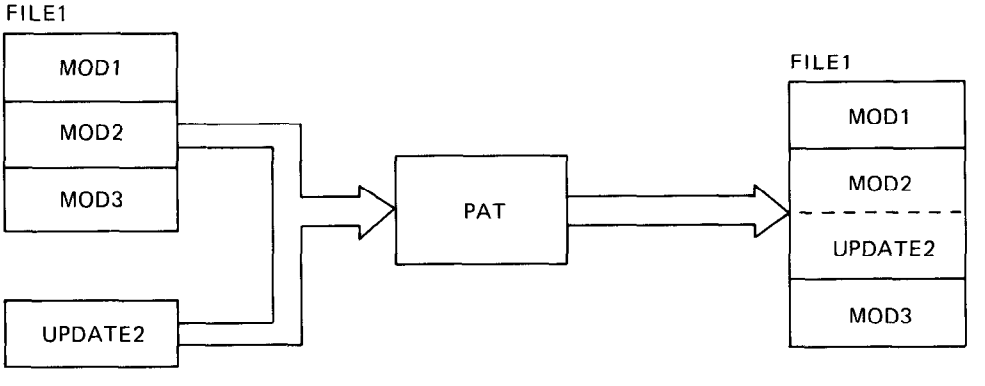


Figure 18-1 Updating a Module Using PAT

There are several steps you must perform to use PAT to update a file. First, create the correction file using a text editor. Then, assemble the correction file to produce an object module. Next, submit the input file and the correction file in object module form to PAT for processing. Finally, link the updated object module, along with the object modules that make up the rest of the file, to resolve global symbols and create an executable program. Figure 18-2 shows the processing steps involved in generating an updated executable file by using PAT.

Specify the PAT command string in the following form:

[output-filespec]=input-filespec[/C[:n]],correct-filespec[/C[:n]]

where:

output-filespec	is the file specification for the output file. If you do not specify an output file, PAT does not generate one.
input-filespec	is the file specification for the input file. This file can contain one or more concatenated object modules.
correct-filespec	is the file specification for the correction file. This file contains the updates being applied to a single module in the input file.
C	specifies the checksum option for the associated file. This directs PAT to generate an octal value for the sum of all the binary data composing the module in that file. (See Section 18.2.5 for more information on checksums.)
n	specifies an octal value. PAT compares the checksum value it computes for a module with the octal value you specify.

18.2 HOW PAT APPLIES UPDATES

PAT applies updates to a base input module by using the additions and corrections you supply in a correction file. This section describes the PAT input and correction files, gives information on how to create the correction file, and gives examples of how to use PAT.

18.2.1 The Input File

The input file is the file to be updated; it is the base for the output file. The input file must be in object module format. When PAT executes, the module in the correction file applies to this file.

18.2.2 The Correction File

The correction file must also be in object module format. It is usually created from a MACRO-11 source file in the following format:

```
.TITLE inputname

.IDENT updatenum

inputline

inputline
    .
    .
    .
```

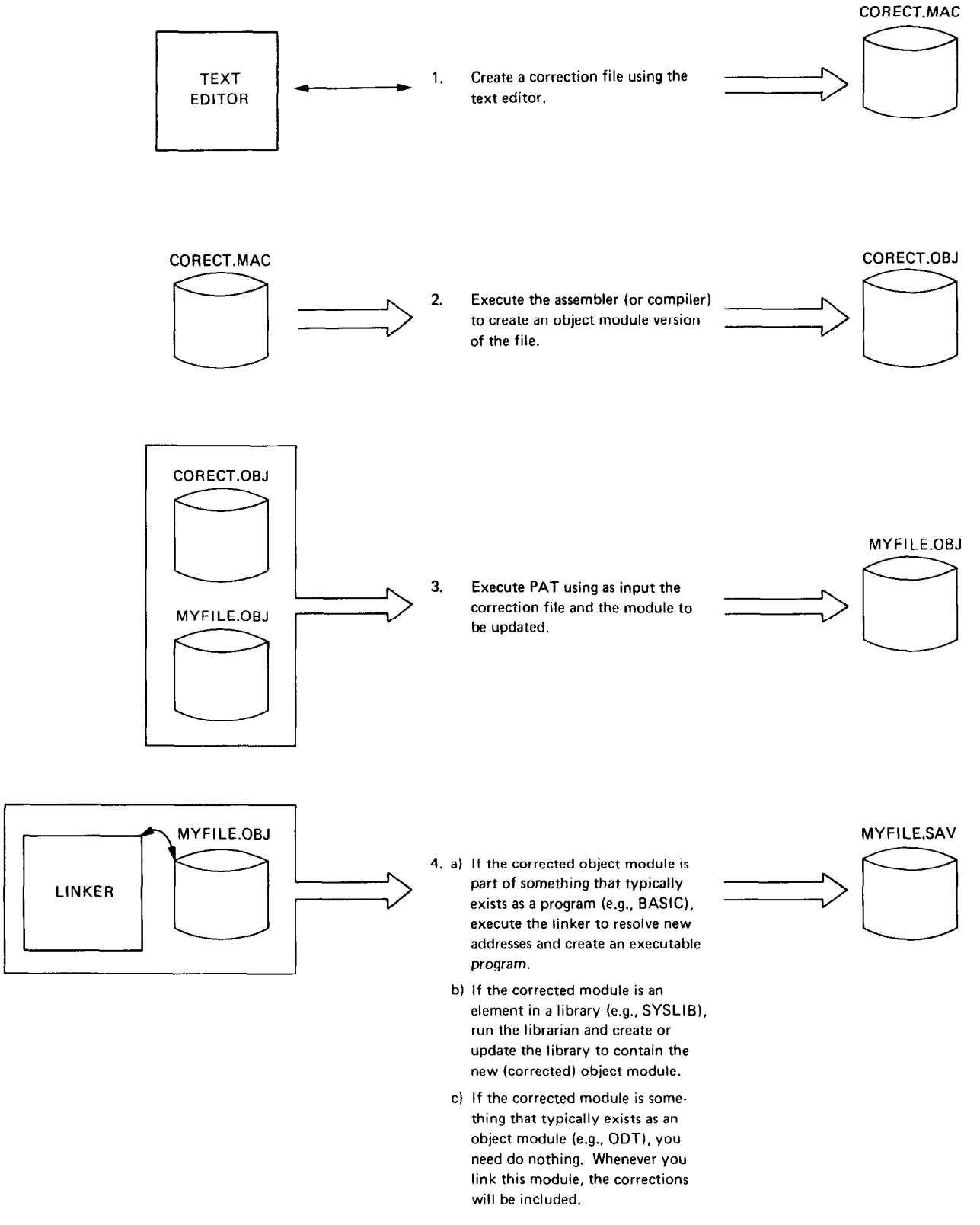


Figure 18-2 Processing Steps Required to Update a Module Using PAT

where:

inputname	is the name of the module to be corrected by the PAT update. That is, inputname must be the same name as the name on the input file .TITLE directive for a single module in the input file.
updatenum	is any value acceptable to the MACRO-11 assembler. Generally, this value reflects the update version of the file being processed by PAT, as shown in the examples below.
inputline	are lines of input for PAT to use to correct and update the input file.

During execution, PAT adds any new global symbols that are defined in the correction file to the module's symbol table. Duplicate global symbols in the correction file supersede their counterparts in the input file, provided both definitions are relocatable or both are absolute.

A duplicate PSECT or CSECT supersedes the previous PSECT or CSECT, provided:

- both have the same relocatability attribute (ABS or REL):
- both are defined with the same directive (.PSECT or .CSECT).

If PAT encounters duplicate PSECT names, it sets the length attribute for the PSECT to the length of the longer PSECT and appends a new PSECT to the module.

If you specify a transfer address, it supersedes that of the module you are patching.

18.2.3 Creating the Correction File

As shown in Figure 18-2, the first step in using PAT to update an object file is to generate the correction file. Use the editor to build a file that contains these additions and corrections. The correction file must be in object module format before PAT can process it. Assemble the correction file with the MACRO-11 assembler to produce an object module that PATCH can process.

18.2.4 How PAT and the Linker Update Object Modules

The following examples show the source code for an input file and a correction file to be processed by PAT and the linker. The examples show as output a single source file that, if assembled and linked, would produce a binary module equivalent to the file generated by PAT and LINK. Two techniques are described: one is for overlaying lines in a module and the other is for appending a subroutine to a module.

18.2.4.1 Overlaying Lines in a Module – The first example illustrates a technique for overlaying lines in a module by using a patch file. First, PAT appends the correction file to the input file. The linker then executes to replace code within the input file.

The source code for the input file for this example is:

```
        .TITLE  ABC
        .IDENT  /01/
        .ENABL  GBL
ABC::
        MOV     A,C
        JSR     PC,XYZ
        RTS     PC
        .END
```

To add the instruction ADD A,B after the JSR instruction, the following patch source file is included:

```
        .TITLE  ABC
        .IDENT  /01.01/
        .ENABL  GBL
.=.+12
        ADD     A,B
        RTS     PC
        .END
```

The patch source is assembled using MACRO-11 and the resulting object file is input to PAT along with the original object file. The following source code represents the result of PAT processing:

```
        .TITLE  ABC
        .IDENT  /01.01/
        .ENABL  GBL
ABC::
        MOV     A,C
        JSR     PC,XYZ
        RTS     PC
.=ABC
.=.12
        ADD     A,B
        RTS     PC
        .END
```

After the linker processes these files, the load image appears as this source-code representation shows:

```
        .TITLE  ABC
        .IDENT  /01.01/
        .ENABL  GBL
ABC::
        MOV     A,C
        JSR     PC,XYZ
        ADD     A,B
        RTS     PC
        .END
```

The linker uses the .=.+12 in the program counter field to determine where to begin overlaying instructions in the program. The linker overlays the RTS instruction with the patch code:

```
        ADD     A,B
        RTS     PC
```

18.2.4.2 Adding a Subroutine to a Module – The second example illustrates a technique for adding a subroutine to an object module. In many cases, a patch requires that more than a few lines be added to patch the file. A convenient technique for adding new code is to append it to the end of the module in the form of a subroutine. This way, you can insert a JSR instruction to the subroutine at an appropriate location. The JSR directs the program to branch to the new code, execute that code, and then return to in-line processing.

The source code for the input file for the example is:

```
        .TITLE  ABC
        .IDENT  /01/
        .ENABL  GBL
ABC::
        MOV     A,B
        JSR     PC,XYZ
        MOV     C,R0
        RTS     PC
        .END
```

Suppose you wish to add the instructions:

```
        MOV     D,R0
        ASL     R0
```

between

```
        MOV     A,B
```

and

```
        JSR     PC,XYZ
```

The correction file to accomplish this goal is as follows:

```
        .TITLE  ABC
        .IDENT  /01.01/
        .ENABL  GBL
        JSR     PC,PATCH
        NOP
        .PSECT  PATCH
PATCH:
        MOV     A,B
        MOV     D,R0
        ASL     R0
        RTS     PC
        .END
```

PAT appends the correction file to the input file, as in the previous example. The linker then processes the file and generates the following output file:

```
        .TITLE  ABC
        .IDENT  /01.01/
        .ENABL  GBL
ABC::
        JSR     PC,PATCH
        NOP
        JSR     PC,XYZ
        MOV     C,R0
        RTS     PC
        .PSECT  PATCH
```

```
PATCH:
        MOV     A,B
        MOV     D,R0
        ASL     R0
        RTS     PC
        .END
```

In this example, the JSR PC,PATCH and NOP instructions overlay the three-word MOV A,B instruction. (The NOP is included because this is a case where a 2-word instruction replaces a 3-word instruction. NOP is required to maintain alignment.) The linker allocates additional storage for .PSECT PATCH, writes the specified code into this program section, and binds the JSR instruction to the first address in this section. Note that the MOV A,B instruction replaced by the JSR PC,PATCH is the first instruction the PATCH subroutine executes.

18.2.5 Determining and Validating the Contents of a File

Use the checksum option (/C) to determine or validate the contents of a module. The checksum option directs PAT to compute the sum of all binary data composing a file. If you specify the command in the form /C:n, /C directs PAT to compute the checksum and compare that checksum to the value you specify with n.

To determine the checksum of a file, enter the PAT command line with the /C option applied to the appropriate file (the file whose checksum you need to determine). For example:

```
*=INFILE/C,INFILE.PAT
```

PAT responds to this command with the message:

```
?PAT-W-Input module checksum is nnnnnn
```

PAT generates a similar message when you request the checksum for the correction file.

To validate the changes made to a file, enter the checksum option in the form /C:n. PAT compares the value it computes for the checksum with the value you specify with n. If the two values do not match, PAT displays a message reporting the checksum error:

```
?PAT-W-Input file checksum error
```

or

```
?PAT-W-Correction file checksum error
```

Checksum processing always results in a nonzero value.

Do not confuse this checksum with the record checksum byte.

APPENDIX A
BATCH

RT-11 BATCH is a complete job control language that allows RT-11 to operate unattended. RT-11 BATCH processing is ideally suited to frequently-run production jobs, large and long-running programs, and programs that require little or no interaction with you, the user. Using BATCH, you can prepare your job on any RT-11 input device and leave it for the operator to start and run.

RT-11 BATCH permits you to:

- Execute an RT-11 BATCH stream from any legal RT-11 input device
- Output a log file to any legal RT-11 output device (except magtape or cassette)
- Execute the BATCH stream with the single-job monitor or in the background with the foreground/background monitor or the extended memory monitor
- Generate and support system-independent BATCH language jobs
- Execute RT-11 monitor commands from the BATCH stream.

RT-11 BATCH consists of 1) the BATCH compiler and 2) the BATCH run-time handler. The BATCH compiler reads the batch input stream you create, translates it into a format suitable for the RT-11 BATCH run-time handler, and stores it in a file. The BATCH run-time handler executes this file with the RT-11 monitor. As each command in the batch stream executes, BATCH lists the command, along with any terminal output generated by executing the command, on the BATCH log device.

A.1 HARDWARE AND SOFTWARE REQUIREMENTS TO RUN BATCH

You can run RT-11 BATCH on any single-job system that is configured with at least 12K words of memory. You need a minimum system of 16K words of memory to run BATCH in the background in a foreground/background environment. BATCH can run in any extended memory environment. A line printer, although optional, is highly desirable as the log device.

BATCH uses certain RT-11 system programs to perform its operations. For example, the $BASIC command executes the file BASIC.SAV. Make sure that the following RT-11 programs are on the system device, with exactly the following names, before you run BATCH:

BASIC.SAV	(BASIC users only)
BA.SYS	
BATCH.SAV	
CREF.SAV	(MACRO users only)
SYSLIB.OBJ	(FORTRAN and MACRO users)
FORTRA.SAV	(FORTRAN users only)
LINK.SAV	
MACRO.SAV	(MACRO users only)
PIP.SAV	
DIR.SAV	

A.2 BATCH CONTROL STATEMENT FORMAT

You can use two forms of input to RT-11 BATCH. Generate a file using the RT-11 editor and input it from any RT-11 input device, or input punched cards from the card reader. In both cases, the input consists of BATCH control statements. A BATCH control statement consists of three fields, separated from one another with spaces: 1) command fields, 2) specification fields, and 3) comment fields. The control statement has the syntax:

```
$command/option      specification/option      [!comment]
```

Each control statement requires a specific combination of command and specification fields and options (see Section A.4). Control statements cannot be longer than 80 characters, excluding multiple spaces, tabs, and comments. You can use a hyphen (-) as a line continuation character to indicate that the control statement is continued on the next line (see Table A-4). Even if you use the line continuation character, the maximum control statement length is still 80 characters.

The following example of a $FORTRAN command illustrates the various fields in a control statement.

```
$FORTRAN/LIST/RUN     PROGA/LIBRARY     PROGB/EXE     !RUN FORTRAN
```

command/options | spec fields/options | comment field

A.2.1 Command Fields

The command field in a BATCH control statement indicates the operation to be performed. It consists of a command name and certain command field options. Indicate the command field with a $ in the first character position and terminate it with a space, tab, blank, or carriage return.

A.2.1.1 Command Names – The command name must appear first in a BATCH control statement. All BATCH command names have a dollar sign ($) in the first position of the command (for example, $JOB). No intervening spaces are allowed in the command name. BATCH recognizes only two forms of a command name: the full name and an abbreviation consisting of $ and the first three characters of the command name. For example, you can enter the $FORTRAN command as:

```
$FORTRAN
```

or

```
$FOR
```

You cannot enter it as:

```
$FORT
```

or

```
$FORTR
```

A.2.1.2 Command Field Options – Options that appear in a command field are command qualifiers. Their functions apply to the entire control statement. All option names must begin with a slash (/) that immediately follows the command name. Table A-1 describes the command field options that are legal in BATCH and indicates the commands on which you can use them. Those option characters that appear in square brackets are optional. These are described in greater detail in the sections pertaining to the commands with which you use them.

NOTE

All /NO options are the defaults, except the /WAIT option in the $MOUNT and $DISMOUNT commands and the /OBJECT option in the $LINK command.

Table A-1 Command Field Options

Option	Explanation
/BAN[NER]	Prints the header of the job on the log file. BATCH allows this option only on the $JOB command.
/NOBAN[NER]	Does not print a job header.
/CRE[F]	Produces a cross reference listing during compilation; BATCH allows this option only on the $MACRO command.
/NOCRE[F]	Does not create a cross reference listing.
/DEL[ETE]	Deletes input files after the operation completes. BATCH allows this option on the $COPY and $PRINT commands.
/NODEL[ETE]	Does not delete input files after operation completes.

/DOL[LARS]	The data following this command can have a $ in the first character position of a line. BATCH allows this option on the $CREATE, $DATA, $FORTRAN, and $MACRO commands. BATCH terminates reading data when you use one of the following commands or when it encounters a physical end-of-file on the BATCH input stream:
	`$JOB`
	`$SEQUENCE`
	`$EOD`
	`$EOJ`

/NODOL[LARS]	Following data cannot have a $ in the first character position; a $ in the first character position signifies a BATCH control command.
/LIB[RARY]	Includes the default library in the link operation. BATCH allows this option on the $LINK and $MACRO commands.
/NOLIB[RARY]	Does not include the default library in the link operation.
/LIS[T]	Produces a temporary listing file (see Section A.2.5) on the listing device (LST:) or writes data images on the log device (LOG:). BATCH allows this option on the $BASIC, $CREATE, $DATA, $FORTRAN, $JOB, and $MACRO commands. When you use /LIST on the $JOB command, /LIST sends data lines in the job stream to the log device (LOG:).
/NOLIS[T]	Does not produce a temporary listing file.
/MAP	Produces a temporary link map on the listing device (LST:). BATCH allows this option on the $FORTRAN, $LINK, and $MACRO commands.

(Continued on next page)

Table A-1 (Cont.) Command Field Options

Option	Explanation
/NOMAP	Does not create a MAP file.
/OBJ[ECT]	Produces a temporary object file as output from compilation or assembly (see Section A.2.5). BATCH allows this option on the $FORTRAN, $LINK, and $MACRO commands. When you use /OBJECT on $LINK, BATCH includes temporary files in the link operation.
/NOOBJ[ECT]	Does not produce an object file as output of compilation; with $LINK, does not include temporary files in the link operation.
/RT11	Sets BATCH to operate in RT-11 mode (see Section A.5). BATCH allows this option only on the $JOB command.
/NORT11	Does not set BATCH to operate in RT-11 mode.
/RUN	Links (if necessary) and executes programs compiled since the last "link-and-go" operation or start of job. BATCH allows this option on the $BASIC, $FORTRAN, $LINK, and $MACRO commands.
/NORUN	Does not execute or link and execute the program after performing the specified command.
/TIM[E]	Writes the time of day to the log file when BATCH executes. BATCH allows this option only on the $JOB command.
/NOTIM[E]	Does not write the time of day to the log file.
/UNI[QUE]	Checks for unique spelling of options and keynames (see Section A.4.13). BATCH allows this option only on the $JOB command.
/NOUNI[QUE]	Does not check for unique spelling.
/WAI[T]	Pauses for operator action. BATCH allows this option on the $DISMOUNT, $MESSAGE, and $MOUNT commands.
/NOWAI[T]	Does not pause for operator action.
/WRI[TE]	Indicates that the operator is to WRITE ENABLE a specified device or volume. BATCH allows this option only on the $MOUNT command.
/NOWRI[TE]	Indicates that no writes are allowed or that the specified volume is read-only; informs the operator, who must WRITE LOCK the appropriate device.

A.2.2 Specification Fields

Specification fields immediately follow command fields in a BATCH control statement. Use them to name the devices and files involved in the command. You must separate these fields from the command field, and from each other, by blanks or spaces.

If a specification field contains more than one file to be used in the same operation, separate the fields by a plus (+) sign. For example, to assemble files F1 and F2 to produce an object file F3 and a temporary listing file, type:

```
$MACRO/LIST F1+F2/SOURCE F3/OBJECT
```

If you need to repeat a command for more than one field specification, separate the files by a comma (,). For example, the following command assembles F1 to produce F2, a temporary listing file, and a map file F3. It then assembles F4 and F5 to produce F6 and a temporary listing file.

```
$MACRO/LIST F1/SOURCE F2/OBJECT F3/MAP,F4+F5/SOURCE-
 F6/OBJECT
```

Note that the command field options apply to the entire line, but the specification field options apply only to the field they follow.

Depending on the command you use, specification fields can contain a device specification, file specification, or an arbitrary ASCII string. You can use an appropriate specification field option (see Table A-3) with any of these three items.

A.2.2.1 Physical Device Names – Represent each device in an RT-11 BATCH specification field with a standard 2- or 3-character device name. Table 3-1 in Chapter 3 lists each name and its related device. If you do not specify a unit number for devices that have more than one unit, BATCH assumes unit 0.

In addition to the permanent names shown in Table 3-1, you can assign logical device names to devices. A logical device name takes precedence over a physical name, thus providing device independence. With this feature, you do not need to rewrite a program that is coded to use a specific device if the device is unavailable. For example, DK: is normally assigned to the system device, but you can assign that name to diskette unit 1 (DX1:) with an RT-11 monitor ASSIGN command.

You must assign certain logical names prior to running any BATCH job. BATCH uses these logical names as default devices. These names are:

LOG:	BATCH log device (cannot be magtape or cassette)
LST:	Default device for listing files generated by BATCH stream.

The following are not legal device names in RT-11; if you use them, the operator must assign them as logical names with the ASSIGN command. You can use these names in BATCH streams written for other DIGITAL systems.

DF:	Fixed-head disk (RF).
LL:	Line printer with upper case and lower case characters.
M7:	7-track magtape.
M9:	9-track magtape.
PS:	Public storage (DK: as assigned by RT-11).

Refer to Sections 4.3 and A.7.1 for instructions on assigning logical names to devices.

A.2.2.2 File Specifications – You can reference files symbolically in a BATCH control statement with a name of up to six alphanumeric characters followed, optionally, by a period and a file type of three alphanumeric characters. Tabs and embedded spaces are not allowed in either the file name or file type. The file type generally indicates the format of a file. It is a good practice to conform to the standard file types for RT-11 BATCH. If you do not specify a file type for an output file, BATCH and most other RT-11 system programs assign appropriate default file types. If you do not specify a file type for an input file, the system searches for that file name with a default file type. Table A-2 lists the standard file types used in RT-11 BATCH.

Table A-2 File Types

File Type	Explanation
.BAS	BASIC source file (BASIC input)
.BAT	BATCH command file
.CTL	BATCH control file generated by the BATCH compiler.
.CTT	BATCH temporary file generated by the BATCH compiler.
.DAT	BASIC or FORTRAN data file
.DIR	Directory listing file
.FOR	FORTRAN IV source file (FORTRAN input)
.LST	Listing file
.LOG	BATCH log file
.MAC	MACRO source file (MACRO or SRCCOM input)
.MAP	Link map output from $LINK operation
.OBJ	Object file output from compilation or assembly
.SOU	Temporary source file
.SAV	$RUNable file or program image output from $LINK

A.2.2.3 Wildcard Construction – You can use wildcards in certain BATCH control statements (such as, $COPY, $CREATE, $DELETE, $DIRECTORY, $PRINT). You can use the asterisk as a wildcard to designate the entire file name or file type. See Chapter 4, Section 4.2, for a complete description of the wild card construction.

NOTE

You cannot use embedded wildcards (* or %) in BATCH control statements. However, you can use them in the keyboard monitor commands if you use the RT-11 mode of BATCH.

A.2.2.4 Specification Field Options – Specification field options follow file specifications in a BATCH control statement and designate how the file is to be used. These options apply only to the field in which they appear. Option names begin with a slash. The specification field options legal in RT-11 BATCH are listed in Table A-3. Optional characters in the option names are in square brackets.

Table A-3 Specification Field Options

Option	Explanation
/BAS[IC]	BASIC source file
/EXE[CUTABLE]	Indicates the executable program image file to be created as the result of a link operation
/FOR[TRAN]	FORTRAN source file
/INP[UT]	Input file; default if you specify no options
/LIB[RARY]	Library file to be included in link operation (prior to default library)
/LIS[T]	Listing file
/LOG[ICAL]	Indicates that the device is a logical device name; use in $DISMOUNT and $MOUNT commands
/MAC[RO]	MACRO source file
/MAP	Linker map file
/OBJ[ECT]	Object file (output of assembly or compilation)
/OUT[PUT]	Output file
/PHY[SICAL]	Indicates physical device name
/SOU[RCE]	Indicates source file
/VID	Volume identification

A.2.3 Comment Fields

Comment fields, which document a BATCH stream, are identified by an exclamation point (!) appearing anywhere except the first character position in the control statement. BATCH treats any character following the ! and preceding the carriage return/line feed combination as a comment. For example:

```
$RUN PIP      !DELETE FILES ON DK:
```

This command runs the RT-11 system program PIP. BATCH ignores the comment.

You can also include comments as separate comment lines by typing a $ in character position 1, followed immediately by the ! operator and the comment. For example:

```
$!DELETE FILES ON DK:
```

A.2.4 BATCH Character Set

The RT-11 BATCH character set is limited to the 64 upper case characters (ASCII 40 through 137). The current ASCII set is assumed (character 137 is underscore and not left arrow, and character 136 is circumflex, not up-arrow). The BATCH job control language does not support any control characters other than tab, carriage return, and line feed.

Table A-4 shows how BATCH normally interprets certain characters. Character interpretations are different if you use RT-11 mode (see Section A.5).

Table A-4 Character Explanation

Character	Explanation
blank/space	Specification field delimiter. It separates arguments in control statements. BATCH considers any string of consecutive spaces and tabs (except in quoted strings) as a blank (that is, equivalent to a single space).
!	Comment delimiter. The input routine ignores all characters after the exclamation point, up to the carriage return/line feed combination.
"	Passes a text string containing delimiting characters where the normal precedence rules would create the wrong action. For example, use it to include a space in a volume identification (/VID).
$	BATCH control statement recognition character. A dollar sign ($) in the first character position of a BATCH input stream line indicates that the line is a control statement.
.	Delimiter for file type.
-	Indicates line continuation if the character after the hyphen is one of the following: • A carriage return/line feed • Any number of spaces or tabs followed by a carriage return/line feed • A comment delimiter (!) • Spaces followed by a comment delimiter (!). If any other character follows the hyphen, the hyphen is assumed to be a minus sign indicating a negative value in an option.
/	Precedes an option name. An alphanumeric string must immediately follow it.
0-9	Numeric string components.
:	Immediately follows a device name. You can also use it to separate an option name from its value or to separate an option value from its subvalue (you can use : interchangeably with = for this purpose).
A-Z	Alphabetic string components.
=	Separates an option name from a value.
\	Illegal character except when it precedes a directive to the BATCH run-time handler from the operator (see Section A.7.3). (To include \ in an RT-11 mode command, use \\.)
+	File delimiter. Separates multiple files in a single specification field. Also indicates a positive value in options.

(Continued on next page)

Table A-4 (Cont.) Character Explanation

Character	Explanation
'	Separates sets of arguments for which the command is to be repeated.
*	A wildcard in utility command file specifications.
CR/LF	Carriage return/line feed. It indicates end-of-line (or end of logical record) for records in the BATCH input stream.

A.2.5 Temporary Files

When you do not include field specifications in a BATCH command line, BATCH sometimes generates temporary files. For example, you can enter a $FORTRAN command that is followed in the BATCH stream by the FORTRAN source program as:

```
$FORTRAN/RUN/OBJECT/LIST
      FORTRAN source program
$EOD
```

This command generates: 1) a temporary source file from the source statements that follow, 2) a temporary object file, 3) a temporary listing file, and 4) a temporary memory image file.

BATCH sends temporary files to the default device (DK:) or the listing device (LST:) according to their nature. If the device is file-structured, BATCH assigns file names and file types as follows:

nnnmmm.LST	for temporary listing files (sent to LST:)
nnnmmm.MAP	for temporary map files (sent to LST:)
nnnppp.OBJ	for temporary object files (sent to DK:)
nnnppp.SAV	for temporary memory image files (sent to DK:)
nnnppp.SOU	for temporary source files (sent to DK:)

where:

nnn	represents the last three digits of the sequence number assigned to the job by the $SEQUENCE command (see Section A.4.22). Thus, a sequence number of 12345 produces a file name beginning 345. If you do not use the $SEQUENCE command, BATCH sets nnn to 000.
mmm	represents the number of listing (or map) files that BATCH generated since the BATCH run-time handler (BA.SYS) was loaded. The first such file, listing or map, is 000. Each time BATCH generates a new temporary file, it increments the file name by 1. Thus, the second listing file produced under job sequence number 12345 is 345001.LST, and the first map file produced is 345000.MAP.
ppp	represents the number of object, memory image, or source files in the current BATCH run. The first such file (object, memory image, or source) is 000. Each time BATCH generates a new temporary file, it increments the file name by 1. BATCH resets these file names to 000 every time that you run BATCH and after every $LINK, $MACRO, or $FORTRAN command that uses the temporaries.

A.3 GENERAL RULES AND CONVENTIONS

You must adhere to the following general rules and conventions associated with RT-11 BATCH processing.

1. Always place a dollar sign ($) in the first character position of a command line.
2. Each job must have a $JOB and $EOJ command (or card).
3. You can spell out command and option names entirely or you can specify only the first three characters of the command and required characters of the option.
4. Specify wildcard construction (*) only for the utility commands ($COPY, $CREATE, $DELETE, $DIRECTORY, and $PRINT) and for commands that normally accept wildcards in RT-11 mode.
5. Include comments at the end of command lines or in a separate comment line. When you include comments in a command line, place them after the command but precede them by an exclamation mark.
6. Include only 80 characters per control statement (card record), excluding multiple spaces, tabs, and comments.
7. When you omit file specifications from BATCH commands and supply data in the BATCH stream, the system creates a temporary file with a default name (see Section A.2.5).
8. You can use the RT-11 monitor type-ahead feature only with BATCH handler directives (see Section A.7.3) to be inserted into a BATCH program. No other terminal input (except input to a foreground program) can be entered while a BATCH stream is executing.
9. You cannot use an indirect command file to call BATCH.

A.4 BATCH COMMANDS

Place BATCH commands in the input stream to indicate to the system which functions to perform in the job. All BATCH commands have a dollar sign ($) in the first character position (e.g., $JOB). Intervening spaces are not allowed in command names. The command name must always start in the first character position of the line (card column 1).

BATCH commands are presented in alphabetical order in this chapter for ease of reference. However, if you are not familiar with BATCH, read the commands in a functional order as listed in Table A-5. The characters shown in square brackets are optional.

Table A-5 BATCH Commands

Command	Section	Explanation
$SEQ[UENCE]	A.4.22	Assigns an arbitrary identification number to a job.
$JOB	A.4.13	Indicates the start of a job.
$EOJ	A.4.11	Indicates the end of a job.
$MOU[NT]	A.4.18	Signals the operator to mount a volume on a device and optionally assigns a logical device name.
$DIS[MOUNT]	A.4.9	Signals the operator to dismount a volume from a device and deassigns a logical device name.
$FOR[TRAN]	A.4.12	Compiles a FORTRAN source program.
$BAS[IC]	A.4.1	Compiles a BASIC source program.
$MAC[RO]	A.4.16	Assembles a MACRO source program.
$LIB[RARY]	A.4.14	Specifies libraries that BATCH should use in link operations.

(Continued on next page)

Table A-5 (Cont.) BATCH Commands

Command	Section	Explanation
$LIN[K]	A.4.15	Links modules for execution.
$RUN	A.4.21	Causes a program to execute.
$CAL[L]	A.4.2	Transfers control to another BATCH file, executes that BATCH file, and returns to the calling BATCH stream.
$CHA[IN]	A.4.3	Passes control to another BATCH file.
$DAT[A]	A.4.6	Indicates the start of data.
$EOD	A.4.10	Indicates the end of data.
$MES[SAGE]	A.4.17	Issues a message to the operator.
$COP[Y]	A.4.4	Copies files.
$CRE[ATE]	A.4.5	Creates new files from data included in the BATCH stream.
$DEL[ETE]	A.4.7	Deletes files.
$DIR[ECTORY]	A.4.8	Provides a directory of the specified device.
$PRI[NT]	A.4.19	Prints files.
$RT[11]	A.4.20	Specifies that the following lines are RT-11 mode commands.

For each command listed below, the term filespec represents a device name, a file name, and a file type.

It has this form:

dev:filnam.typ

As a general rule, BATCH assumes device DK: if you omit a device specification.

A.4.1 $BASIC Command

The $BASIC command calls RT-11 single-user BASIC to execute a BASIC source program. The $BASIC command has the following syntax:

$BASIC[/option . . .] [filespec/option]] [!comments]

where:

/option — indicates an option you can append to the $BASIC command. The options are as follows:

- /RUN — indicates that BATCH should execute the source program.
- /NORUN — indicates that BATCH should only compile the program, and send error messages to the log file.

/LIST writes data images that are contained in the job stream to the log file (LOG:).

/NOLIST writes data images to the log file only if you specify $JOB/LIST.

filespec indicates the name and type of the source file and the device on which it resides. If you omit the file type, BATCH assumes .BAS. If you omit this specification, the source statements must immediately follow the $BASIC command in the input stream.

Terminate the source program after a $BASIC statement with either a $EOD command or with any other BATCH command that starts with a $ in the first position.

/option indicates an option that can follow the source file name. BATCH assumes that any file name with no option appended is the name of a source file. This option can have one of the following values (or you can omit it):

/BASIC indicates that the file name you specify is a BASIC source program.

/SOURCE performs the same function as /BASIC.

/INPUT performs the same function as /BASIC.

You can follow the $BASIC command with the source program, legal BASIC commands (such as RUN), or data. The following two BATCH streams, for example, produce the same results.

```
$BASIC                    $BASIC/RUN
10 INPUT A                10 INPUT A
20 PRINT A                20 PRINT A
30 END                    30 END
RUN                       $DATA
123                       123
$EOD                      $EOD
```

A.4.2 $CALL Command

The $CALL command transfers control to another BATCH control file, temporarily suspending execution of the current control file. BATCH executes the called file until it reaches $EOJ or until the job aborts; control then returns to the statement following the $CALL in the originating BATCH control file. You can nest calls up to 31 levels. BATCH includes the log file for the called file in the log file for the originating BATCH program. (See NOTE following the $EOJ command.)

The syntax of the $CALL command is:

$CALL filespec[!comments]

BATCH does not permit options in the $CALL command. BATCH saves $JOB command options across a $CALL; however, they do not apply to the called BATCH file. If you specify .CTL as the file type, BATCH assumes a precompiled BATCH control file. If you do not specify a file type, BATCH assumes .BAT and compiles the called BATCH stream before execution.

NOTE

If the called program generates temporary files, those files can supersede currently existing temporary files if the two jobs have the same sequence number. For example, consider the following two BATCH streams:

```
$FOR/OBJ A          $FOR/OBJ A
$FOR/OBJ B          $CALL C
$LINK/RUN           $FOR/OBJ B
                    $LINK/RUN
```

The called BATCH file (C.BAT) contains the following:

```
$JOB
$FOR/OBJ A1
$FOR/OBJ B1
$LINK/RUN
$EOJ
```

The temporary object files that C.BAT generates change the behavior of the previous two BATCH statement sequences. The first temporary file created by C.BAT (000000.OBJ) supersedes the temporary file produced by the first $FORTRAN command (000000.OBJ). Avoid this situation by giving the BATCH job C.BAT a unique sequence number (see Section A.4.22).

A.4.3 $CHAIN Command

The $CHAIN command transfers control to a named BATCH control file but does not return to the input stream that executed the $CHAIN command. The syntax of the $CHAIN command is:

$CHAIN filespec[!comments]

BATCH does not permit options in the $CHAIN command. If you specify .CTL as the file type, BATCH assumes a precompiled BATCH control file. If you do not specify a file type, BATCH assumes .BAT and compiles the chained BATCH stream before execution.

A $EOJ command should always follow the $CHAIN command in the BATCH stream.

NOTE

The values of BATCH run-time variables remain constant across a $CALL, $CHAIN, or return from call. See Section A.5.2.2 for a description of these variables.

Use the $CHAIN command to transfer control to programs that you need to run only once at the end of a BATCH stream. For example, you could use the following BATCH program (PRINT.BAT) to print and then delete all temporary listing files generated during the current BATCH job.

```
$JOB                          !PRINT ALL LIST FILES
$PRINT/DELETE *.LST
$EOJ
```

You could then run PRINT.BAT with the $CHAIN command as follows:

```
$JOB
$MACRO/RUN   A   ALST/LIST
$MACRO/RUN   B   BLST/LIST
$CHAIN PRINT
$EOJ
```

A.4.4 $COPY Command

The $COPY command copies files in image mode from one device to another. You can use the wildcard construction (see Section A.2.2.3) in the input and output file specifications. You can concatenate several input files to form one output file (as long as the output specification does not contain a wildcard). The $COPY command has the following syntax:

```
$COPY[/option] output-filespec [ . . . , output-filespec]/OUTPUT-
   input-filespec[ . . . , input-filespec] [/INPUT] [!comments]
```

where:

/option	indicates options that you can append to the $COPY command.
	/DELETE deletes input files after the copy operation.
	/NODELETE does not delete input files after the copy operation.
output-filespec	represents an output file. You must specify a file type.
/OUTPUT	indicates that a file specification is for an output file.
input-filespec	represents a file to be copied.
	BATCH copies files to the output file in the order that you list them (except when you use wildcards).
/INPUT	indicates that a file specification is for an input file. If you do not specify an option, BATCH assumes INPUT.

The following are examples of the $COPY command:

```
$COPY *.BAS/OUTPUT DT1:*.BAS
```

This command copies all files with the file type .BAS from the DECtape on unit 1 to the default storage device DK:.

```
$COPY FILE2.FOR/OUTPUT FILE0.FOR+FILE1.FOR
```

This command merges the input files FILE0.FOR and FILE1.FOR to form one file called FILE2.FOR and stores FILE2.FOR on device DK:.

```
$COPY *.*/OUT DT0:*.FOR, DT1:*.*/OUT DT0:*.*
```

This command copies all files with the file type .FOR from DT0: to DK: and all files on DT0: to DT1:.

A.4.5 $CREATE Command

The $CREATE command generates a file from data records that follow the $CREATE command in the input stream. An error occurs if the data does not immediately follow the $CREATE command. You cannot precede the data records with a $DATA command.

You can follow the $CREATE data with a $EOD command to signify the end of data, or you can use any other BATCH control statement to indicate end of data and initiate a new function. The $CREATE command has the following syntax:

```
$CREATE[/option . . .] filespec [!comments]
```

where:

/option	indicates an option you can append to the $CREATE command. The options are:
	/DOLLARS — indicates that the data following this command can have a $ in the first character position of a line.
	/NODOLLARS — indicates that a $ cannot be in the first character position of a line.
	/LIST — writes data image lines to the log file.
	/NOLIST — does not write data image lines to the log file. If you specify $JOB/LIST, BATCH ignores this option.
filespec	represents the file you want to create.

NOTE

If you use the /DOLLARS option, you must follow the last data record with a $EOD command (see Table A-1).

The following is an example of the $CREATE command:

```
$CREATE/LIST PROG.FOR
     FORTRAN source file
$EOD
```

The data records following the $CREATE command become a new file (PROG.FOR) on the default device (DK:). BATCH generates a listing on logical device LOG:.

A.4.6 $DATA Command

Use the $DATA command to include data records in the input stream. Data you include in this manner needs no file name. BATCH transfers the data to the appropriate program as though it were input from the console terminal. For example, you can follow the $RUN command for a particular program by a $DATA command and the data records for the program to process. The data records must be valid data for the program that is to use them.

The $DATA command has the following syntax:

```
$DATA[/option . . .] [!comments]
```

Four options that you can use with the $DATA command are as follows:

/DOLLARS	indicates that the data following this command can have a $ in the first character position of a line.

/NODOLLARS	indicates that a $ cannot be in the first character position of a line.
/LIST	writes data image lines to the log file.
/NOLIST	does not write data images to the log file. If you specify $JOB/LIST, BATCH ignores this option.

NOTE
Any command beginning with a $ normally follows the last data record. However, if you specify $DATA/DOLLARS, you must follow the last data record with $EOD.

The following example shows data entered into a BASIC program (TEST1.BAS).

```
$BASIC/RUN TEST1.BAS
$DATA
25,75,125,146
180,210,520,874
$EOD
```

A.4.6.1 Using $DATA with FORTRAN Programs – When you use the $DATA command to provide input to a FORTRAN program, you must insert a CTRL/Z into the BATCH file after the last data line and before $EOD (or before the next BATCH command if you do not use $EOD). This procedure permits FORTRAN to properly detect an end-of-file after it reads the last data line. For example:

```
$FORTRAN/RUN  A.FOR
$DATA
1
2
3
^Z (RET) (LF)
$EOD
$RUN PIP
```

The above program reads three numbers from the input stream and then detects an end-of-file when it attempts to read a fourth number. If you include an END=n statement in your FORTRAN program, statement n gets control when the end-of-file is detected. If the CTRL/Z (RET) (LF) is not present, the program aborts when it reaches $EOD and never executes the END=n statement.

A.4.7 $DELETE Command

Use the $DELETE command to delete files from the device you specify. This command has the syntax:

$DELETE filespec [. . . , filespec] [!comments]

where:

filespec	represents the name of a file to be deleted.

The following example deletes all files named TEST1 on the default device DK:.

```
$DELETE TEST1.*
```

The following example deletes all files with .FOR file types on DT1:, then deletes all files with .MAC file types on DK:.

```
$DELETE DT1:*.FOR,*.MAC
```

A.4.8 $DIRECTORY Command

The $DIRECTORY command outputs a directory of the device you specify to a listing file. If you do not specify a listing file, the listing goes to the BATCH log file. This command has the syntax:

$DIRECTORY [filespec/LIST] [filespec[. . . , filespec]] [/INPUT] [!comments]

where:

filespec/LIST	indicates the name of the directory listing file.
filespec/INPUT	indicates the input files to be included in the directory (default).

The following are examples of the $DIRECTORY command:

```
$DIRECTORY
```

This command outputs a directory of the device DK: to the BATCH log file.

```
$DIRECTORY FOR.DIR/LIST *.FOR
```

This command creates a directory file (FOR.DIR) on the device DK:. The directory contains the names, lengths, and dates of creation of all FORTRAN source files on the device DK:.

A.4.9 $DISMOUNT Command

The $DISMOUNT command removes the logical device name assigned by a $MOUNT command. When BATCH encounters $DISMOUNT while executing a job, it prints the entire $DISMOUNT command line on the console terminal. This message tells the operator which device to unload. This command has the syntax:

$DISMOUNT[/option] logical-device-name:[/LOGICAL] [!comments]

where:

/option	indicates an option you can append to the $DISMOUNT command. The options are:	
	/WAIT	indicates that the job must pause until the operator enters a response. If you do not specify either /WAIT or /NOWAIT, BATCH assumes /WAIT. BATCH rings a bell at the terminal, prints the physical device name to be dismounted followed by a question mark (?), and waits for a response. (At this point you can enter input to the BATCH handler. See Section A.7.3.)
	/NOWAIT	does not pause for operator response, BATCH prints the physical device name to be dismounted.
logical-device-name:	is the logical device name to be deassigned from the physical device.	
/LOGICAL	identifies the device specification as a logical device name.	

The following example instructs the operator to dismount the physical device with the logical device name OUT: and removes the logical assignment of device OUT:. In this example, OUT: is DT0:. The operator dismounts DT0: and then types a carriage return.

```
$DISMOUNT/WAIT OUT:/LOGICAL
DT0?
```

A.4.10 $EOD Command

The $EOD command indicates the end-of-data record or the end of a source program in the job stream. The syntax of this command is:

```
$EOD [!comments]
```

The $EOD command can signal the end of data associated with any of the following commands:

```
$BASIC
$CREATE
$DATA
$FORTRAN
$MACRO
```

In the following example, the $EOD command indicates the end of a source program that is to be compiled, linked, and executed.

```
$FORTRAN/RUN
      source program
$EOD
```

A.4.11 $EOJ Command

The $EOJ command indicates the end of a job. This command must be the last statement in every BATCH job. The command has the following syntax:

```
$EOJ [!comments]
```

If BATCH encounters a $JOB command, a $SEQUENCE command, or a physical end-of-file in the input stream before $EOJ, an error message appears in the log file.

NOTE

Make sure that the $EOJ command is the last line in a .BAT file.

A.4.12 $FORTRAN Command

The $FORTRAN command calls the FORTRAN compiler to compile a source program. Optionally, this command can provide printed listings or list files and can produce a link map in the listing. The $FORTRAN command has the following syntax:

```
$FORTRAN[/option . . .] [source-filespec[/option]] [filespec/OBJECT] [filespec/LIST] -
    [filespec/EXECUTE] [filespec/MAP] [filespec/LIBRARY] [!comments]
```

where:

/option	indicates an option you can append to the $FORTRAN command. The options are as follows:

	/RUN	indicates that FORTRAN is to compile the source program, link it with the default library, and execute it. The default library is SYSLIB.OBJ. You can change it with the $LIBRARY command.
	/NORUN	compiles the program only.
	/OBJECT	produces a temporary object file.
	/NOOBJECT	does not produce a temporary object file.
	/LIST	produces a list file on the listing device (LST:).
	/NOLIST	does not produce a list file.
	/MAP	produces a link map on the listing device (LST:).
	/NOMAP	does not create a MAP file.
	/DOLLARS	indicates that the data following this command can have a $ in the first character position of a line.
	/NODOLLARS	indicates that a $ cannot be in the first character position of a line.

source-filespec — indicates the device, file name, and file type of the FORTRAN source file. If you do not specify the file name, the $FORTRAN source statements must immediately follow the $FORTRAN command in the input stream; BATCH generates a temporary source file that it deletes after FORTRAN compiles the temporary source file (see Section A.2.5).

You can terminate the source program included after a $FORTRAN statement by either a $EOD command or by any other BATCH command. If, however, you use dollar signs in the first position in the source program, you must enter the source program with $CREATE/DOLLARS. In this case, you cannot use $FORTRAN/DOLLARS.

/option — represents an option that can have one of the following values:

/FORTRAN	indicates that the file name you specify is a FORTRAN source program. BATCH assumes that any file name with no option appended is the name of a source file.
/SOURCE	performs the same function as /FORTRAN.
/INPUT	performs the same function as /FORTRAN.

filespec/OBJECT — indicates the device, file name, and file type of the object file produced by compilation. The object file remains on the device you specify after the job finishes. You must follow the object file specification, if you include it, by the /OBJECT option.

If you omit the object file specification but specify $FORTRAN/OBJECT, BATCH creates a temporary object file. BATCH includes this temporary file in any $LINK operations that follow it in the job, and deletes it after the link operation.

filespec/LIST	indicates the name you assign to the list file created by the compiler. BATCH does not automatically print the list file if you assign LST: to a file-structured device, but you can list it using the $PRINT command. Follow the list file specification by the /LIST option.
filespec/EXECUTE	indicates the name you assign to a memory image file. Follow the memory image file specification by the /EXECUTE option. If you do not include this field, BATCH generates a temporary memory image file (see Section A.2.5) and then deletes the temporary file.
filespec/MAP	indicates the name you assign to the link map file created by the linker. Follow the map specification by the /MAP option.
filespec/LIBRARY	indicates that BATCH must include the file you specify in the link procedure as a library before SYSLIB.OBJ. The file must be a library file (produced by the RT-11 librarian). Follow the library specification by the /LIBRARY option.

The following are examples of $FORTRAN commands:

```
$FORTRAN/RUN PROGA.FOR
```

This command calls FORTRAN to compile and execute a source program named PROGA.FOR.

```
$FORTRAN/NOOBJ/LIST
     source program
$EOD
```

This command sequence compiles the FORTRAN program but does not produce an object file. BATCH creates a temporary listing file on LST:.

NOTE

See Section A.4.6.1 for instructions on using the $DATA command with FORTRAN programs.

A.4.13 $JOB Command

The $JOB command indicates the beginning of a job. Each job must have its own $JOB command. This command has the following syntax:

$JOB[/option . . .] [!comments]

BATCH allows the following options in the $JOB command:

/BANNER	prints a header (a repetition of the $JOB command) on the log file.
/NOBANNER	does not print a job header.
/LIST	writes data image lines that are contained in the job stream to the log file.
/NOLIST	writes data image lines to the log file only when a /LIST option exists on a $BASIC, $CREATE, or $DATA command that has data lines following it.
/RT11	if no $ appears in column 1 when BATCH expects one, BATCH assumes that the line or card is an RT-11 mode command (see Section A.5).

/NORT11	does not process RT-11 mode commands.
/TIME	writes the time of day to the log file when BATCH executes command lines (except $DATA command lines).
/NOTIME	does not write the time of day.
/UNIQUE	checks for unique spelling of options and keynames. When you use this option, you can abbreviate commands and options to the least number of characters that still make their names unique. For example, you can abbreviate the /DOLLARS option to /DO since no other option begins with the characters DO.
/NOUNIQUE	checks only for normal option and keyname spellings.

End each job with a $EOJ command if you want to run it. If an input stream consists of more than one job, BATCH automatically terminates one job when it encounters the $JOB command for the next job. BATCH will never run a job terminated with another $JOB command; an error message will appear in the log.

The following $JOB command writes the time of day to the log file before BATCH executes each command beginning with a $. It also accepts unique abbreviations of BATCH commands and options.

```
$JOB/TIME/UNIQUE
```

A.4.14 $LIBRARY Command

The $LIBRARY command lets you specify a list of library files that will be included in FORTRAN links or with other link operations that have the /LIBRARY option. By default, the list of libraries contains only SYSLIB.OBJ, the RT-11 system library. This command has the syntax:

```
$LIBRARY filespec [!comments]
```

or

```
$LIBRARY filespec+SYSLIB [!comments]
```

where:

filespec	represents a library file; the default file type is .OBJ.
SYSLIB	is the RT-11 system library that you create at system generation.

Libraries are linked in order of their appearance in the $LIBRARY command.

The following example shows two libraries (LIB1.OBJ and LIB2.OBJ) that are included in FORTRAN links before SYSLIB.OBJ.

```
$LIBRARY LIB1.OBJ+LIB2.OBJ+SYSLIB.OBJ
```

A.4.15 $LINK Command

Use the $LINK command to produce memory image files from object files. This command links the files you specify (if any) with all temporary object files created since the last link or link-and-go operation (if any).

Temporary object files are those files you create as a result of a $FORTRAN or $MACRO command without naming an object file (with the /OBJECT option) or suppressing an object file (with the /NOOBJECT option). Create permanent object files by using the /OBJECT option on a $FORTRAN or $MACRO file descriptor.

BATCH links files in the following order:

1. First, it links temporary files in the order in which they were compiled.
2. Then, it links permanent files in the order in which they are specified in the $LINK command.
3. If the $LINK command specifies a library, BATCH links it next, providing that unresolved references remain.
4. If you specify $LINK/LIBRARY, BATCH searches and links the default library list.

The syntax for this command is:

```
$LINK[/option . . . ] [filespec/OBJECT] [filespec/LIBRARY] [filespec/MAP] [filespec/EXECUTE] -
    [!comments]
```

where:

/option	indicates an option that you can append to the $LINK command. The options are as follows:
/LIBRARY	includes the RT-11 system library (SYSLIB.OBJ) and any default libraries specified in the $LIBRARY command in this $LINK operation. Use this option when the files being linked do not include any temporary FORTRAN object files. You can also use it when you specify $FORTRAN without the /RUN or /MAP option, but you want to search the default library list for unresolved references.
/NOLIBRARY	does not include the default libraries.
/MAP	produces a temporary load map on the listing device (LST:).
/NOMAP	does not produce a map file.
/OBJECT	includes temporary object files in the link. If you specify neither /OBJECT nor /NOOBJECT, BATCH assumes $LINK/OBJECT.
/NOOBJECT	does not include temporary files in the link.
/RUN	executes the memory image files associated with this $LINK command when the link is complete.
/NORUN	only links the program and does not execute it.
filespec/OBJECT	indicates the name of the object file BATCH must link. If you do not specify /OBJECT, BATCH assumes it as the default.
filespec/LIBRARY	indicates that the file you specify is to be included in the link procedure as a library. The file you specify must be a library file (produced by the RT-11 librarian).
filespec/MAP	indicates the load map file BATCH must create as a result of the $LINK command.
filespec/EXECUTE	indicates the memory image file BATCH must create as a result of the $LINK command.

The following are examples of the $LINK command:

```
$LINK/RUN
```

This command links all temporary object files created since the last $LINK command or the last $FORTRAN/OBJ or $MACRO/OBJ command.

```
$LINK/MAP PROG1.OBJ+PROG2.OBJ/OBJ PROGA.SAV/EXE
```

This command links the temporary files and the object files PROG1.OBJ and PROG2.OBJ to form a memory image file named PROGA.SAV. It also creates and outputs a temporary map file.

A.4.16 $MACRO Command

The $MACRO command calls the MACRO assembler to assemble a source program and, optionally, to provide printed listings or list files. You must specify MACRO listing directives, if any, in the source program. You cannot enter them at BATCH command level.

The $MACRO command has the following syntax:

```
$MACRO[/option . . .] [source-filespec[/option]] [filespec/OBJECT] [filespec/LIST] -
    [filespec/MAP] [filespec/LIBRARY] [filespec/EXECUTE] [!comments]
```

where:

/option	indicates an option you can append to the $MACRO command. The options are as follows:	
	/RUN	assembles, links, and runs the source program.
	/NORUN	only assembles the source program.
	/OBJECT	produces a temporary object file.
	/NOOBJECT	does not produce a temporary object file.
	/LIST	produces a listing file on the listing device (LST:).
	/NOLIST	does not produce a list file.
	/CREF	produces a cross reference listing during assembly.
	/NOCREF	does not produce a cross reference listing during assembly.
	/MAP	produces a link map as part of the listing file on LST:.
	/NOMAP	does not create a MAP file.
	/DOLLARS	indicates that the data following this command can have a $ in the first character position of a line.
	/NODOLLARS	indicates that a $ cannot be in the first character position of a line.
	/LIBRARY	includes the default library (SYSLIB.OBJ) in the link operation.

/NOLIBRARY does not include the default library in the link operation.

source-filespec indicates the name of the source file. If you do not specify a file name, the $MACRO source statements must immediately follow the $MACRO command in the input stream.

You can terminate the source program you include after a $MACRO statement by either a $EOD command or by any other BATCH command. If, however, you include dollar signs in the first position in the source program, you must use the $CREATE/DOLLARS command to enter the source program. In this case, you cannot use $MACRO/DOLLARS.

/option can have one of the following values:

/MACRO indicates that the file name you specify is a MACRO source program. BATCH assumes that any file name with no option appended is the name of a source file.

/SOURCE performs the same function as /MACRO.

/INPUT performs the same function as /MACRO.

filespec/OBJECT indicates the name you assign to the object file produced by compilation. The object file remains on the device you specify after the job finishes. If you include an object file specification, follow it with the /OBJECT option.

If you omit the object file specification but specify $MACRO/OBJECT, BATCH creates a temporary object file. BATCH also includes the temporary object file in any $LINK operations that follow the $MACRO command in the job, and deletes it after the link operation (see Section A.2.5).

filespec/LIST indicates the name you assign to the list file created by the assembler. BATCH does not automatically print the list file if you assign LST: to a file-structured device, but you can list it using the $PRINT command. The /LIST option must follow the list file specification.

filespec/MAP indicates the file to which BATCH must output the storage map.

filespec/LIBRARY indicates that BATCH must include the file you specify in the link procedure as a library. The /LIBRARY option must follow the library file specification.

filespec/EXECUTE indicates the name you assign to a memory image file. The /EXECUTE option must follow the memory image file specification. If you do not include this field but do use $MACRO/RUN, BATCH generates a temporary memory image file (see Section A.2.5) and runs it.

The following $MACRO command assembles a program named PROG0.MAC and creates a temporary object file and a temporary listing file.

```
$MACRO/LIST/OBJECT PROG0.MAC
```

A.4.17 $MESSAGE Command

Use the $MESSAGE command to issue a message to the operator at the console terminal. It provides a means for the job to communicate with the operator. The $MESSAGE command has the syntax:

```
$MESSAGE[/option] message [!comments]
```

where:

/option	indicates an option you can append to the $MESSAGE command. The options are:
	/WAIT — indicates that the job is to pause until the operator either types a carriage return to continue or enters commands to the BATCH handler followed by a carriage return (see Section A.7.3).
	/NOWAIT — does not pause for operator response.
message	is a string of characters that must fit on one console line. BATCH prints the message on the console.

For example, if you include the following message in the input stream:

```
$MESSAGE/WAIT MOUNT SCRATCH TAPE ON MT0:
```

The message:

```
MOUNT SCRATCH TAPE ON MT0:
?
```

appears on the console terminal and a bell sounds. The operator mounts the tape and types carriage return to allow further processing of the job. (See Section A.7.3 for operator interaction with BATCH.)

NOTE

BATCH compresses multiple spaces and tabs in BATCH command lines; therefore, attempts to format $MESSAGE output with tabs or spaces do not provide you with the desired results.

A.4.18 $MOUNT Command

The $MOUNT command assigns a logical device name and other characteristics to a physical device. When BATCH encounters $MOUNT during the execution of a job, it prints the entire $MOUNT command line on the console terminal to notify the operator which volume to use.

The $MOUNT command has the syntax:

```
$MOUNT[/option . . . ] physical-device-name: [/PHYSICAL] [/VID=x]
    [logical-device-name:/LOGICAL] [!comments]
```

where:

/option	indicates an option you can append to the $MOUNT command. The options are:
	/WAIT — indicates that the job is to pause until the operator enters a response. If you do not specify either /WAIT or /NOWAIT, BATCH assumes /WAIT. BATCH rings a bell, prints the physical device name and a

question mark (?), and waits for a response. (The response can consist of input for the BATCH handler; see Section A.7.3.)

/NOWAIT — does not pause for operator response. BATCH prints the name of the physical device to be mounted.

/WRITE — tells the operator to WRITE ENABLE the volume.

/NOWRITE — tells the operator to WRITE PROTECT the volume.

physical-device-name — is required and specifies the physical device name and an optional unit number followed by a colon (for example, DX1:). If you specify a device name without a unit number, the operator can enter one in response to the question mark printed by the $MOUNT command. If you want the operator to supply a unit number, do not use the /NOWAIT option because it assumes unit 0.

/PHYSICAL — identifies the device specification as a physical unit specification. If you do not specify either /PHYSICAL or /LOGICAL, BATCH assumes /PHYSICAL.

/VID=x
/VID="x" — provides volume identification. The volume identification is the name physically attached to the volume. Include it to help the operator locate the volume. Use this option only on the physical device file specification. If x contains spaces, specify it as "x".

NOTE

This volume identification is only a visual check for the operator. Make this volume identification match the visual label on the volume, not the volume identification that you wrote onto the volume at initialization time with the INITIALIZE/VOLUMEID command.

logical-device-name/LOGICAL — is required to identify the logical device name, if any, you assign to the device. The /LOGICAL option must follow the logical device name specification.

The following command instructs the operator to select a DECtape unit and mount DECtape volume BAT01 on that unit, WRITE ENABLED. It informs the operator by printing:

```
$MOUNT/WAIT/WRITE DT:/VID=BAT01 2:/LOGICAL
DT?
```

The operator selects a unit, mounts DECtape volume BAT01, WRITE ENABLED, and responds to the question mark by typing the unit number (such as 1) followed by a carriage return. BATCH assigns logical device name 2 to the physical device (in this case DT1:) and proceeds.

If no unit number response is necessary, as this command shows,

```
$MOUNT/WAIT/WRITE DT1: 2:/LOGICAL
DT1?
```

the operator responds with a carriage return after mounting the DECtape and WRITE ENABLING the device.

A.4.19 $PRINT Command

Use the $PRINT command to print the contents of the files you specify on the listing device (LST:). This command has the syntax:

```
$PRINT[/option] filespec [. . . ,filespec] [/INPUT] [!comments]
```

where:

/option	indicates an option you can append to the $PRINT command. The options are:
	/DELETE deletes input files after printing.
	/NODELETE does not delete input files after printing.
filespec	represents a file to be printed.
/INPUT	indicates that the file is an input file; BATCH assumes /INPUT if you omit it.

The following command prints a listing of files with file type .MAC that are stored on default device DK:.

```
$PRINT *.MAC
```

The following example creates listing files for the programs A and B, prints the listing files, and then deletes them.

```
$MACRO A.MAC A/LIST
$MACRO B.MAC B/LIST
$PRINT/DELETE A.LST,B.LST
```

A.4.20 $RT11 Command

The $RT11 command allows the BATCH job to communicate directly with the RT-11 system. DIGITAL recommends that you use RT-11 mode if you use BATCH. This command puts BATCH in RT-11 mode until BATCH encounters a line beginning with $. In RT-11 mode, BATCH interprets all data images as commands to the RT-11 monitor, to RT-11 system programs, or to the BATCH run-time system. The $RT11 command has the syntax:

```
$RT11[!comments]
```

See Section A.5 for a complete description of the RT-11 mode.

A.4.21 $RUN Command

The $RUN command executes a program for which a memory image file (.SAV) was previously created. It can also run RT-11 system programs.

The $RUN command has the syntax:

```
$RUN filespec[!comments]
```

where:

filespec	represents the file to be executed. If you omit the file type, BATCH assumes .SAV.

For example, you can run DIR to print a directory listing:

```
$RUN DIR
$DATA
LP:=DK:/L
$EOD
```

A.4.22 $SEQUENCE Command

The $SEQUENCE command is an optional command. If you use it, it must immediately precede a $JOB command. The $SEQUENCE command assigns a job an arbitrary identification number. BATCH assigns the last three characters of a sequence number as the first three characters of a temporary listing or object file (see Section A.2.5). If a sequence number is less than three characters long, BATCH fills it with zeroes on the left.

The syntax of this command is:

$SEQUENCE id [!comments]

where:

id	represents an unsigned decimal number that indicates the identification number of a job.

The following are examples of the $SEQUENCE command:

```
$SEQUENCE 3             !SEQUENCE NUMBER IS 003
$JOB

$SEQUENCE 100           !SEQUENCE NUMBER IS 100
$JOB
```

A.4.23 Sample BATCH Stream

The following sample BATCH stream creates a MACRO program, assembles and links that program, and runs the memory image file. It then deletes the object, memory image, and source files it created and prints a directory of DK: showing the files the BATCH stream created.

```
$JOB
$MESSAGE          THIS IS AN EXAMPLE BATCH STREAM
$MESSAGE          NOW CREATE A MACRO PROGRAM
$CREATE/LIST      EXAMPL.MAC
.TITLE   EXAMPL FOR BATCH
         .MCALL   .PRINT,.EXIT
START:   .PRINT   #MESSAG
         .EXIT
MESSAG:  .ASCIZ   /EXAMPLE MACRO PROGRAM FOR BATCH/
         .END     START
$EOD
$MACRO   EXAMPL EXAMPL/OBJECT EXAMPL/LIST    !ASSEMBLE
$LINK    EXAMPL EXAMPL/EXECUTE               !AND LINK
$PRINT/DELETE EXAMPL.LST
$MESSAGE          RUN THE MACRO PROGRAM
$RUN     EXAMPL                              !AND EXECUTE
$DELETE EXAMPL.OBJ+EXAMPL.SAV+EXAMPL.MAC
$MESSAGE          PRINT A DIRECTORY
$DIRECTORY        DK:EXAMPL.*
$MESSAGE          END OF THE EXAMPLE BATCH STREAM
$EOJ
```

To run this batch stream, type the following commands at the console. BATCH prints the messages.

```
.LOAD BA,LP
.ASSIGN LP: LOG
.ASSIGN LP: LST
.R BATCH
*EXAMPL
 THIS IS AN EXAMPLE BATCH STREAM
 NOW CREATE A MACRO PROGRAM
 RUN THE MACRO PROGRAM
 PRINT A DIRECTORY
 END OF THE EXAMPLE BATCH STREAM

END BATCH
.
```

The above sample BATCH stream produces the following log file on the line printer:

NOTE

The amount of free core and the directory format are variable.

```
$JOB

$MESSAGE        THIS IS AN EXAMPLE BATCH STREAM

$MESSAGE        NOW CREATE A MACRO PROG.

$CREATE/LIST    EXAMPL.MAC

.TITLE  EXAMPLE FOR BATCH
        .MCALL  .PRINT,.EXIT
START:  .PRINT  #MESSAG
        .EXIT
MESSAG: .ASCIZ  /EXAMPLE MACRO PROGRAM FOR BATCH/
        .EVEN
        .END    START
0

$EOD

$MACRO  EXAMPL EXAMPL/OBJECT EXAMPL/LIST    !ASSEMBLE

ERRORS DETECTED:  0

EXAMPLE FOR BATCH       MACRO V03.00 21-JUN-77 00:05:29 PAGE 1

      1                                 .TITLE  EXAMPLE FOR BATCH
      2                                         .MCALL  .PRINT,.EXIT
      3 000000                          START:  .PRINT  #MESSAG
      4 000006                                  .EXIT
      5 000010     105     130     101  MESSAG: .ASCIZ  /EXAMPLE MACRO PROGRAM FOR BATC
        000013     115     120     114
        000016     105     040     115
        000021     101     103     122
        000024     117     040     120
        000027     122     117     107
        000032     122     101     115
        000035     040     106     117
        000040     122     040     102
        000043     101     124     103
        000046     110     000
      6                                 .EVEN
      7        000000'                  .END    START
```

```
EXAMPLE FOR BATCH        MACRO V03.00 21-JUN-77 00:05:29 PAGE 1-1
SYMBOL TABLE

MESSAG  000010R          START   000000R

. ABS.  000000     000
        000050     001
ERRORS DETECTED:  0

VIRTUAL MEMORY USED:  508 WORDS  ( 2 PAGES)
DYNAMIC MEMORY AVAILABLE FOR  48 PAGES
EXAMPL,EXAMPL=EXAMPL

$LINK   EXAMPL EXAMPL/EXECUTE              !AND LINK

$PRINT/DELETE EXAMPL.LST


$MESSAGE        RUN THE MACRO PROGRAM

$RUN    EXAMPL                                    !AND EXECUTE

EXAMPLE MACRO PROGRAM FOR BATCH

$DELETE EXAMPL.OBJ+EXAMPL.SAV+EXAMPL.MAC


$MESSAGE        PRINT A DIRECTORY

$DIRECTORY      DK:EXAMPL.*

 21-JUN-77
EXAMPL.BAK     2 14-JUN-77       EXAMPL.BAT     2 21-JUN-77
EXAMPL.CTL     3 21-JUN-77
 3 FILES, 7 BLOCKS
 1903 FREE BLOCKS

$MESSAGE        END OF THE EXAMPLE BATCH STREAM

$EOJ
```

A.5 RT-11 MODE

RT-11 mode lets you enter commands to the RT-11 monitor or to system programs, and lets you create BATCH programs. You can enter RT-11 mode with either the $JOB/RT11 command or the $RT11 command. If you enter RT-11 mode with the $JOB/RT11 command, RT-11 mode remains in effect until BATCH encounters the next command. If you enter RT-11 mode with the $RT11 command, RT-11 mode is in effect until BATCH encounters a $ in the first position of the command line.

The characters . , $, *, and tab or space appearing in the first position of a line (or card column 1) are control characters and indicate the following:

. command to the RT-11 monitor, such as

```
.R PIP
```

* data line; any line not intended to go to the RT-11 monitor or to the BATCH run handler, such as a command to the RT-11 PIP program:

```
*FILE1.DAT/D
```

NOTE

BATCH does not pass the * as data to the program. Comment lines (!) cannot appear on data lines as BATCH would consider them as data.

$ BATCH command. It causes an exit from RT-11 mode if you entered RT-11 mode with the $RT11 command. For example:

```
$RT11                          !ENTER RT-11 MODE
.R PIP
*FILE1.DAT/D
$FORTRAN                       !LEAVE RT-11 MODE
```

space/tab separator to indicate a line directed to the BATCH run-time handler. This separator is indicated by a (TAB) in the following descriptions.

A.5.1 Communicating with RT-11

The most common use of RT-11 mode is to send commands to the RT-11 monitor and to run system programs. For example, you can insert the following commands in the BATCH stream to run PIP and save backup copies of files on DECtape:

```
$RT11
.R PIP
*DT1:*.*=*.FOR
```

You must anticipate and include in the BATCH input stream responses that the called program requires, such as the Y response to DUP's ARE YOU SURE? query. Place a line in your BATCH file consisting of Y and RETURN or use the DUP /Y option to suppress the query. For example:

```
$RT11
.INITIALIZE RK1:
*Y
```

You can communicate directly with the RT-11 monitor by using the keyboard monitor commands that are described in Section 4.3. For example:

```
$RT11
.DELETE/NOQUERY DX1:*.MAC
```

This command deletes all files with a file type of .MAC from device DX1:.

You cannot mix BATCH standard commands with RT-11 mode data lines (lines beginning with an asterisk). For example, the proper way to do a $MOUNT within a sequence of RT-11 mode data commands is:

```
$JOB/RT11
.R MACRO
*A1=A1
*A2=A2
$MOUNT DT0:/PHYSICAL
.R MACRO
*B1=DT:B1
*B2=DT:B2
```

A.5.2 Creating RT-11 Mode BATCH Programs

Advanced system programmers can use RT-11 mode to create BATCH programs. These BATCH programs consist of standard RT-11 mode commands (monitor commands, data lines for input to system programs, etc.) plus special RT-11 mode commands. The BATCH run-time handler interprets these special commands to allow dynamic calculations and conditional execution of the RT-11 mode standard commands. The following can help you create BATCH programs and dynamically control their execution at run-time:

- Labels
- Variable modification:

  1) equating a variable to a constant or character (LET statement)
  2) incrementing the value of a variable by 1
  3) reading a value into a variable
  4) conditional transfers on comparison of variable values with numeric or character values (IF and GOTO statements)

- Commands to control terminal I/O
- Other Control Characters
- Comments

A.5.2.1 Labels – You define labels in RT-11 mode to provide a symbolic means of referring to a specific location within a BATCH program. If present, a label must begin in the first character position, must be unique within the first six characters, and must terminate with a colon (:) and a carriage return/line feed combination.

A.5.2.2 Variables – A variable in RT-11 mode is a symbol representing a value that can change during program execution. The 26 variables BATCH permits in a BATCH program have the names A-Z; each variable requires one byte of physical storage. You can assign values to variables in a LET statement. You can then test these values by an IF statement to control the direction of program execution.

Assign values to variables with a LET statement of the following form:

(TAB) LET x="c

where:

x	represents a variable name in the range A-Z.
"c	indicates the ASCII value of a character.

For example:

```
(TAB) LET     A="0
```

This example indicates that the value of variable A is the 7-bit ASCII value of the character 0 (60).

The LET statement can also specify an octal value in the form:

(TAB) LET A=n

where:

n	represents an 8-bit signed octal value in the range 0-377. Positive numbers range from 0-177; negative numbers range from 200-377 (-200 to -1).

You can use variables to introduce control characters, such as ESCAPE, into a BATCH stream. For example, wherever 'A' appears in the following BATCH stream, BATCH substitutes the contents of variable A (the code for an ESCAPE):

```
$JOB/RT11
   LET   A=33
   !A  IS  AN  ESCAPE
.R EDIT
*EBFILE,MAC'A"A'
*R'A"A'
   !EDIT FILE TO CHANGE THE VERSION NUMBER TO 2
*GVERSION='A'DI2"A'
*EX'A"A'
```

Increment the value of a variable by 1 by placing a percent sign (%) before the variable. For example:

```
(TAB) %A
```

This command indicates that BATCH must increase the unsigned contents of variable A by 1.

Indicate with an IF statement conditional transfers of control according to the value of a variable. The IF statement has the syntax:

(TAB) IF(x-"c) label1, label2, label3

or

(TAB) IF(x-n) label1, label2, label3

where:

x	represents the variable to be tested.
"c	is the ASCII value to be compared with the contents of the variable.
n	is an octal integer in the range 0-377.
label1 label2 label3	represent the names of labels included in the BATCH stream.

When BATCH evaluates the expression (x-"c) or (x-n), the BATCH run-time handler transfers control to:

- label1 if the value of the expression is less than zero.
- label2 if the value of the expression is equal to zero.
- label3 if the value of the expression is greater than zero.

If you omit one of the labels, and the condition is met for the omitted label, control transfers to the line following the IF statement.

NOTE

Since this comparison is a signed byte comparison, 377 is considered to be -1.

The characters + and - allow you to control where BATCH begins searching for label1, label2, and label3. If you precede the label by a minus sign (-), BATCH starts the label search just after the $JOB command. If a plus sign (+) or no sign precedes the label, the label search starts after the IF statement. For example:

```
TAB IF(B-"9) -LOOP, LOOP1,
```

This statement transfers program control to the label LOOP following the $JOB command if the contents of variable B are less than the ASCII value of 9. It transfers control to the label LOOP1 following the IF statement if B is equal to ASCII 9. If the contents of variable B are greater than the ASCII value of 9, program control goes to the next BATCH statement in sequence.

The GOTO statement unconditionally transfers program control to a label you specify as the argument of the statement. You can use one of the following three forms of this statement:

TAB GOTO label	transfers control to the first occurrence of label that appears after this GOTO statement in the BATCH stream.
TAB GOTO +label	same as GOTO label.
TAB GOTO -label	transfers control to the first occurrence of label that appears after the $JOB command.

The following GOTO statement transfers control unconditionally to the next label LOOP if such a label appears in the BATCH stream following the GOTO statement.

```
TAB GOTO  LOOP
```

NOTE

If BATCH cannot find a label (for example, if you unintentionally omit a minus sign) the BATCH handler searches until it reaches the end of the .CTL file and ends the job.

A.5.2.3 Terminal I/O Control – You can issue commands directly to the BATCH run-time handler to control logging console terminal input and output. If you do not enter any of the following commands, BATCH assumes TTYOUT.

TAB NOTTY	does not write terminal input and output to the log file. Comments to the log are still logged.
TAB TTYIN	writes only terminal input to the log file.
TAB TTYIO	writes terminal input and output to the log file. (You should enter this command if you are using RT-11 mode so that RT-11 mode commands go to the log file.)
TAB TTYOUT	writes only terminal output to the log file (default).

A.5.2.4 Other Control Characters – The system permits other control characters in an RT-11 mode command that begins with a period (.) or an asterisk (*). Following are these control characters and their meanings:

'text'	command to BATCH run-time handler, where text can be one of the following:	
	CTY	accepts input from the console terminal; notifies the operator that action is required by ringing a bell and printing a question mark (?).
	FF	outputs the current log buffer.

NL	inserts a new line (line feed) in the BATCH stream.
x	inserts the contents of a variable where x is an alphanumeric variable in the range A through Z. It indicates that BATCH should insert the contents of the variable as an ASCII character at this place in the command string.
"message"	directs the message to the console terminal.

The following commands allow the operator to enter the name of a MACRO program to be assembled. The BATCH stream contains:

```
$JOB/RT11
.R MACRO
*'"ENTER MACRO COMMAND STRING"''CTY'
```

The operator receives the following message at the terminal and types a response, followed by carriage return; BATCH processing continues.

```
ENTER MACRO COMMAND STRING
?FILE,FILE=FILE
```

To run the same BATCH file on several systems with different configurations you need to assign a device dynamically. The following RT-11 mode command lets you request that the listing device name be entered by the operator.

```
.ASSIGN '"PLEASE TYPE LST DEVICE NAME"''CTY' LST
```

The operator receives the message and responds with the device to be used as the listing device (DT2:).

```
PLEASE TYPE LST DEVICE NAME
?DT2:
```

A.5.2.5 Comments – You can include comments in RT-11 mode as separate comment statements. Include comments by typing a separator followed by a ! and the comment. For example:

```
(TAB)!OPERATOR ACTION IS REQUESTED IN THIS JOB. BE PREPARED.
```

A.5.3 RT-11 Mode Examples

The following are examples of BATCH programs using the RT-11 mode.

This BATCH program assembles, lists, and maps 10 programs with only 12 BATCH commands.

```
$JOB/RT11     !ASSEMBLE, LIST, MAP PROG0 to PROG9
   TTYIO
   !WRITE TERMINAL I/O TO THE LOG FILE
   LET N="0
   !START AT FILE PROG0
LOOP:
.R MACRO
*PROG'N',LOG:/C=PROG'N'/N:TTM
.R LINK
*,LOG:=PROG'N'
   %N
   !INCREMENT VARIABLE N
   IF(N-"9)-LOOP,-LOOP,END
   !TEST FOR END
END:
$EOJ
```

The following program lets you set up a master control stream to run several BATCH jobs with one call to BATCH. First set up a BATCH job (INIT.BAT) that performs a $CHAIN to the master control stream:

```
$JOB/RT11
   LET I="0
   !INITIALIZE INDEX
$CHAIN MASTER          !GO TO MASTER
$EOJ
```

The following is the master control stream (MASTER.BAT) to which INIT chains.

```
$JOB/RT11              !MASTER CONTROL STREAM
   %I
   !BUMP INDEX BY 1
   IF(I-"7),,END
.R BATCH
   !THIS IS A $CHAIN
*JOB'I'
   !RUNS JOB1-JOB7
END:
$MESSAGE END OF BATCH RUN
$EOJ
```

Each job that MASTER.BAT runs must contain the following:

```
$JOB
   !BATCH COMMANDS
$CHAIN MASTER
$EOJ
```

Activate the master control stream by calling BATCH as follows:

```
.R BATCH
*INIT
```

A.6 CREATING BATCH PROGRAMS ON PUNCHED CARDS

To create a BATCH program on punched cards, punch into the cards the commands described in Section A.4. Each command line occupies a single punched card. Only one card, the EOF card, is different from the standard BATCH commands. The EOF (end-of-file) card terminates the list of jobs from the card reader.

To create the EOF card, hold the MULT PCH key on the keypunch keyboard while typing the following characters:

```
- & 0 1 6 7 8 9
```

This procedure produces an EOF card with holes punched in the first column (see Figure A-1).

To run multiple jobs from the card reader, simply combine the jobs into a single card deck. Ensure that each job has its own $JOB and $EOJ card. Then follow the last $EOJ card with two EOF cards.

Although in general, you terminate BATCH jobs on cards by placing two EOF cards after the last $EOJ card, some card readers require that you type \F followed by a carriage return. Put two EOF cards and a blank card in the reader and ensure that the card reader is ready. Note that a small card deck (less than 512 characters) can require more than two EOF cards to terminate the deck.

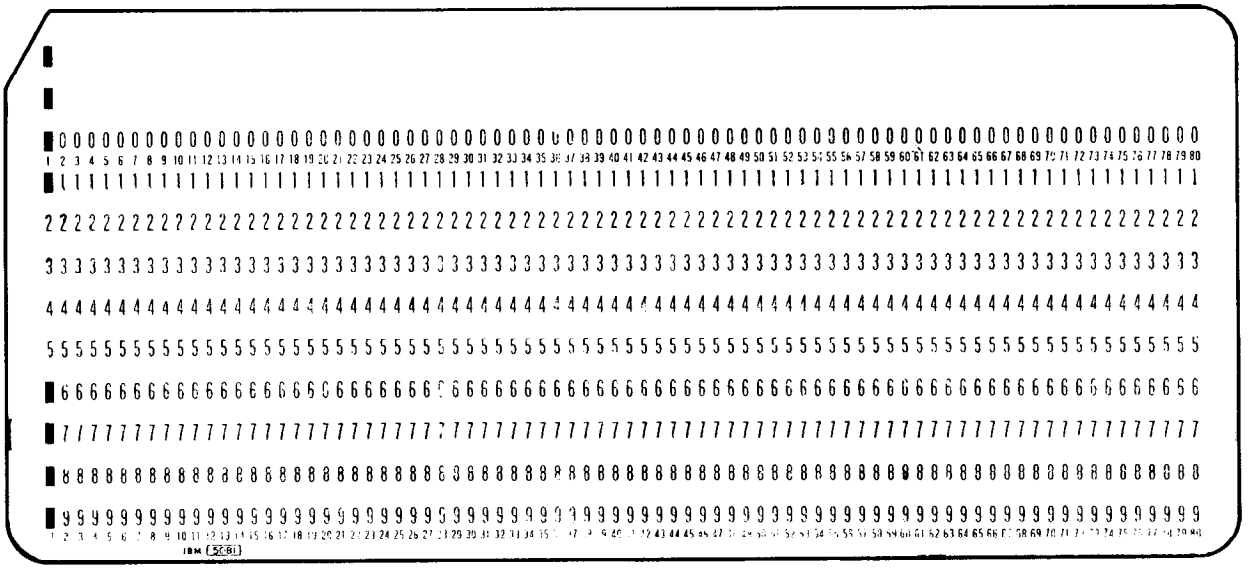


Figure A-1 EOF Card

A.7 OPERATING PROCEDURES

A.7.1 Loading BATCH

After you bootstrap the RT-11 system and enter the date and time, you must make the BATCH run-time handler resident by typing the RT-11 LOAD command as follows:

```
.LOAD BA:
```

You detach and unload the BATCH run-time handler with the /U option in the BATCH compiler command line (see Section A.7.2).

NOTE

If BATCH crashes, you must unload BATCH with the UNLOAD command and then reload BATCH with the LOAD command. This ensures that the BATCH handler is properly initialized when you rerun BATCH.

You must make the BATCH log device resident unless the log device is SY:, or unless it is a device for which the handler is already resident. Load the log device by typing:

```
.LOAD log
```

where:

log	represents the device to which BATCH must write the log file.

For example:

```
.LOAD LP:
```

You can, of course, load device handlers with a single LOAD command. For example:

```
.LOAD BA:,LP:
```

You must then assign the logical device name LOG to the log device. Use the RT-11 monitor ASSIGN command in the form:

.ASSIGN log LOG

For example, if LP: is the log device, type:

```
.ASSIGN LP: LOG
```

Then assign the logical device name LST: using the RT-11 ASSIGN command in the form:

.ASSIGN list-device LST

where:

list-device	represents the physical device BATCH must use for listings.

If, for example, you want to produce listings on the line printer, type:

```
.ASSIGN LP: LST
```

NOTE

Do not use the DEASSIGN command with no arguments in a BATCH program since it deassigns the log and list devices, possibly causing the BATCH job to terminate.

You must also make resident the BATCH run-time handler input device (compiler output device). If this device is already resident or is SY:, you do not need to load it. For example, to load the DECtape handler as the input device, type:

```
.LOAD DT:
```

If the input file to the BATCH compiler is on cards, load the card reader handler by typing:

```
.LOAD CR:
```

NOTE

If input is on cards, you must use the RT-11 monitor SET command (before loading the handler) to specify CRLF and NOIMAGE modes. That is, the following command appends a carriage return/line feed combination to each card image.

```
.SET CR: CRLF
```

The following command translates the card by packing card code into ASCII data, one column per byte.

```
.SET CR: NOIMAGE
```

If card images do not properly translate to ASCII, you may have to change the card translation codes by using one of the following commands:

```
.SET CR: CODE=29
```

or

```
.SET CR: CODE=26
```

See Section 4.4.

A.7.2 Running BATCH

When you have loaded all necessary handlers, run the BATCH compiler as follows:

```
.R BATCH
```

BATCH responds by printing an asterisk (*) to indicate its readiness to accept commands. In response to the *, type the output file specifications for the control file followed by an equal sign. Then type the input file specifications for the BATCH file as follows:

[[output-filespec] [,log-filespec] [/option . . .] =] input-filespec [. . . , input-filespec] [/option . . .]

where:

output-filespec — is the BATCH compiler output device and file the BATCH run-time handler must use. The device you specify must be random-access. Your BATCH job should not delete or move this file. Your BATCH job should avoid compressing the system volume with the SQUEEZE command or the DUP /S option. If you omit the output-filespec, BATCH generates a file on the default device DK: with the same name as the first input file but with a .CTL file type. If you do not specify a file type in the output-filespec, BATCH assumes .CTL.

log-filespec — is the log file created by the BATCH run-time handler. If you do not specify a log device, BATCH assumes LOG:. The device name you specify for log-filespec must be the same as you assign to LOG:.

You can change the size of a log file on a file-structured device from the default size of 64 (decimal) blocks. To make this change, enclose the required size in square brackets. For example:

```
*,FILE.LOG[10]=FILE
```

The default file type for the log-filespec is .LOG.

input-filespec — represents an input file. If you do not specify a file type, BATCH assumes .BAT. If you specify a .CTL file, BATCH assumes a precompiled file that must be the only file in the input list.

/option — is an option from the following list:

/N — compiles but does not execute. This option creates a BATCH control file (.CTL), generates an ABORT JOB message at the beginning of the log file, and returns to the RT-11 monitor.

/T:n — if n=0, sets the /NOTIME option as the default on the $JOB command. If n=1, the default option on the $JOB command is /TIME.

/U — indicates that the BATCH compiler must detach the BATCH run-time handler from the RT-11 monitor and unload the handler.

NOTE

You need not specify the RT-11 monitor UNLOAD BA command to actually remove the handler. Specifying /U to BATCH causes the handler to detach and unload.

/X	indicates that the input is a precompiled BATCH program. Use this option when you do not specify the .CTL file type.
(RET)	prints the version number of the BATCH compiler.

The following example calls BATCH to compile and execute the three input files (PROG1.BAT, PROG2.BAT, PROG3.BAT) to generate on DK: the compiler output files, and to generate on LOG: a log file.

```
.R BATCH
*PROG1.BAT,PROG2.BAT,PROG3.BAT
```

The following commands print the version number of BATCH, then compile and run SYBILD.BAT.

```
.R BATCH
*(RET)
BATCH V03.02
*SYBILD
```

The following commands compile PROTO.BAT to create PROTO.CTL but do not run the compiled program.

```
.R BATCH
*PROTO/N
```

Type the following commands to unlink BA.SYS from the monitor and to unload it.

```
.R BATCH
*/U
```

The following commands compile FILE.BAT from magtape to create FILE.CTL on RK1:. They execute the compiled file and create a log file named FILE.LOG (of size 20) on LOG:.

```
.R BATCH
*RK1:FILE,FILE[20]=MT:FILE
```

The following commands execute a precompiled job called FILE.TST.

```
.R BATCH
*FILE.TST/X
```

The following commands execute a precompiled job called FILE.CTL.

```
.R BATCH
*FILE/X
```

The following commands accept input from the card reader to create a file called TEMP.CTL. BATCH stores this file on DK: and executes it.

```
.R BATCH
*CR:
```

The following commands accept input from the card reader to create a file called JOB.CTL. BATCH stores the file on DK: and executes it.

```
.R BATCH
*JOB=CR:
```

A.7.3 Communicating with BATCH Jobs

During the execution of a BATCH stream, BATCH can request the operator to service a peripheral device, to provide information, or to insert a command line into the BATCH stream. The operator does this by typing directives to the BATCH handler on the console terminal.

NOTE

These directives are equivalent to the compiler output that BATCH generates in the .CTL file. The .CTL file is an ASCII file that you can list by using the PRINT or TYPE commands or by running PIP.

These directives have the form:

```
\dir
```

where:

dir — represents one of the directives listed in Table A-6.

To use these directives, the operator must get control of the BATCH run-time handler by typing a carriage return on the console terminal. When BATCH executes a command, it acknowledges the carriage return and prints a carriage return/line feed combination at the terminal. The operator can then enter a directive from Table A-6. The most useful directives are marked with an asterisk (*).

Table A-6 Operator Directives to BATCH Run-Time Handler

Directive	Explanation
*\A	Changes the input source to be the console terminal.
*\B	Changes the input source to be the BATCH stream.
*\C	Sends the following characters to the log device.
*\D	Considers the following characters as user data.
*\E	Sends the following characters to the RT-11 monitor.
*\F	Forces the output of the current log block. If this directive is followed by any characters other than another BATCH backslash (\) directive, the BATCH job prints an error message and terminates. BATCH then returns control to the RT-11 monitor.
\Hn	Is the help function to change the logging mode; n specifies the following: 0 logs only .TTYOUT and .PRINT. 1 logs .TTYOUT, .PRINT, and .TTYIN 2 does not log .TTYOUT, .PRINT, and .TTYIN 3 logs only .TTYIN

In this example, the operator must interrupt the BATCH handler to enter information from the console. As a result of a /WAIT or 'CTY' in the BATCH stream, the following message appears at the terminal:

```
$MESSAGE/WAIT WRITE NECESSARY FILES TO DISK
```

To divert BATCH stream input from the current file to the console terminal, the operator types a \E, enters commands to the RT-11 monitor, then types a \B. Control then returns to the BATCH stream. The following example illustrates this procedure.

```
.R BATCH
*NEXT
 WRITE NECESSARY FILES TO DISK
?\A\E

\ECOPY DT1:FILE.MAC RK:

 FILES COPIED:
DT1:FILE.MAC   TO RK:FILE.MAC

\E\F\B


END BATCH
```

The following BATCH program lets you make frequent edits to a file and list only the edits. First, create a BATCH program that assembles with a listing and then link the file. This BATCH program, called COMPIL.BAT, contains:

```
$JOB/RT11
   TTYIO
   !WRITE TERMINAL I/O TO LOG FILE
.R MACRO
   !CALL THE MACRO ASSEMBLER
*FILE,FILE/C=FILE
$MESSAGE/WAIT OK TO TYPE EDIT COMMANDS
.R LINK
   !CALL THE RT-11 LINKER
*FILE,LOG:=FILE
$EOJ
```

At run-time, you can insert commands into the BATCH stream from the console terminal. These commands search for the section of the listing file that has been edited then lists this section to the log. You must insert the command after the R MACRO command but before the R LINK command. The following example illustrates this procedure.

```
.R BATCH
*COMPIL
 OK TO TYPE EDIT COMMANDS
?\A\E

\ER EDIT

*ERFILE.LST$$
*EWFILE.SEC$$
*PRETRY:$=J$$
*\L$$
RETRY:  0                          ;HIGH ORDER BIT USED FOR "RESET IN PROGRESS FLAG
     49 000020  016705  177764                  MOV     RKCQE,R5          ;GET Q P
     50 000024  011502                          MOV     @R5,R2            ;R2 = BL
     51 000026  016504  000002                  MOV     2(R5),R4          ;R4 = UN
     52 000032  006204                          ASR     R4                ;ISOLATE
     53 000034  006204                          ASR     R4
     54 000036  006204                          ASR     R4
     55 000040  000304                          SWAB    R4
```

```
     56 000042  042704  017777                BIC     #^C<160000>,R4
     57 000046  000404                        BR      2$              ;ENTER C
*EX$$

\E\C\B

END BATCH
```

A.7.4 Terminating BATCH

When BATCH terminates normally, it prints the following message and returns control to the RT-11 monitor:

```
END BATCH
```

To abort BATCH while it is executing a BATCH stream, interrupt the BATCH handler by typing a carriage return. When BATCH executes the next command after the carriage return, it prints a carriage return/line feed combination at the console terminal. You then gain control of the system. Type \F followed by a carriage return. The BATCH handler responds with the FE (forced exit) error message and writes the remainder of the log buffer. Control returns to the RT-11 monitor.

Typing two CTRL/Cs interrupts and terminates BATCH immediately. Use two CTRL/Cs when BATCH is in a loop or when a long assembly is running. In these cases, BATCH does not respond promptly (or at all) to your carriage return interrupt.

A.8 DIFFERENCES BETWEEN RT-11 BATCH AND RSX-11D BATCH

Some programmers run their RT-11 BATCH programs under RSX-11D. Note the differences between the two BATCH implementations listed in Table A-7. BATCH programs that run under both systems must be compatible with both RT-11 and RSX-11D BATCH.

Table A-7 Differences Between RT-11 and RSX-11D BATCH

Characteristic	RT-11	RSX-11D
File descriptors	filespec/option	SY:filnam.typ/option
Default listing file type	.LST(or .LIS)	.LIS
Executable file type	.SAV	.EXE
Incompatible commands	$BASIC $CALL $CHAIN $LIBRARY $RT11 $SEQUENCE	$MCR
Incompatible options	$COPY/DELETE $CREATE/DOLLARS $CREATE/LIST $DATA/DOLLARS $DATA/LIST $DIR file/LIST $DISMOUNT/WAIT $DISMOUNT lun:/LOGICAL $FORTRAN/DOLLARS $FORTRAN/MAP	$DIR file/DIRECTORY

(Continued on next page)

Table A-7 (Cont.) Differences Between RT-11 and RSX-11D BATCH

Characteristic	RT-11	RSX-11D
Incompatible options (Cont.)	$JOB/BANNER $JOB/LIST $JOB/RT11 $JOB/TIME $JOB/UNIQUE $LINK/LIBRARY $LINK/OBJECT $MACRO/CREF $MACRO/DOLLARS $MACRO/LIBRARY $MACRO/MAP $MESSAGE/WAIT $MESSAGE/WRITE $PRINT/DELETE	$JOB/NAME $JOB/LIMIT $JOB/MCR $LINK/MCR
$DATA input	appears as if from input	appears as if from a file named FOR001.DAT
Logical device names	in $MOUNT and $DISMOUNT	logical unit numbers only
$RUN	you must specify file name	RSX11DBAT.EXE is default

APPENDIX B
MONITOR COMMAND ABBREVIATIONS AND SYSTEM PROGRAM EQUIVALENTS

This appendix provides a table of correspondence (Table B-1) between the keyboard monitor commands with their options and the system utility programs with their options. Remember that the syntax you use to issue a keyboard monitor command is different from the syntax that the Command String Interpreter requires for input and output specifications for the system utility programs. Bear in mind that there are many differences between issuing a monitor command and running a utility program. Table B-1 lists all the keyboard monitor commands and options. A dash under the corresponding system program or option column indicates that the command has no real system program equivalent, that the function is inherent in the keyboard monitor, or that the function is the default mode of operation. The minimum abbreviation for each command and option is underlined.

Table B-1 Monitor Command/System Program Equivalents

Monitor Command	Option	System Utility Program	Option
APL		R APL	–
ASSIGN		–	–
B		–	–
BASIC		R BASIC	–
BOOT		DUP	/O
CLOSE		–	–
COMPILE		–	–
	/ALLOCATE:size	–	[n]
	/ALPHABETIZE	DIBOL	/A
	/CODE:type	FORTRAN	/I
	/CROSSREFERENCE[:type[. . . :type]]	MACRO, DIBOL	/C
	/DIAGNOSE	FORTRAN	/B
	/DIBOL	–	–
	/DISABLE:value[. . . :value]	MACRO	/D
	/ENABLE:value[. . . :value]	MACRO	/E
	/EXTEND	FORTRAN	/E
	/FORTRAN	–	–
	/HEADER	FORTRAN	/O
	/I4	FORTRAN	/T
	/LIBRARY	MACRO	/M
	/LINENUMBERS	–	–
	/NOLINENUMBERS	DIBOL, FORTRAN	/O /S
	/LIST[:filespec]	–	–
	/MACRO	–	–
	/OBJECT[:filespec]	–	–
	/NOOBJECT	–	–
	/ONDEBUG	DIBOL, FORTRAN	/D
	/OPTIMIZE[:type]	FORTRAN	/P
	/NOOPTIMIZE[:type]	FORTRAN	/M
	/PASS:1	MACRO	/P

(Continued on next page)

Table B-1 Monitor Command/System Program Equivalents (Cont.)

Monitor Command	Option	System Utility Program	Option
	/RECORD:length	FORTRAN	/R
	/SHOW:value	FORTRAN, MACRO	/L
	/NOSHOW:value	MACRO	/N
	/STATISTICS	FORTRAN	/A
	/SWAP	–	–
	/NOSWAP	FORTRAN	/U
	/UNITS:n	FORTRAN	/N
	/VECTORS	–	–
	/NOVECTORS	FORTRAN	/V
	/WARNINGS	FORTRAN	/W
	/NOWARNINGS	DIBOL	/W
COPY		–	–
	/ALLOCATE:size	–	[n]
	/ASCII	PIP	/A
	/BINARY	PIP	/B
	/BOOT	DUP	/U
	/CONCATENATE	PIP	/U
	/DEVICE	DUP	/I
	/DOS	FILEX	/S
	/EXCLUDE	PIP	/P
	/IGNORE	PIP	/G
	/IMAGE	–	–
	/INTERCHANGE[:size]	FILEX	/U
	/LOG	PIP	/W
	/NOLOG	–	–
	/NEWFILES	PIP	/C
	/OWNER:[nnn, nnn]	FILEX	UIC
	/PACKED	FILEX	/P
	/POSITION:n	PIP	/M
	/PREDELETE	PIP	/O
	/QUERY	PIP	/Q
	/NOQUERY	–	–
	/REPLACE	–	–
	/NOREPLACE	PIP	/N
	/SETDATE	PIP	/T
	/SLOWLY	PIP	/S
	/SYSTEM	PIP	/Y
	/TOPS	FILEX	/T
D		–	–
DATE		–	–
DEASSIGN		–	–
DELETE		–	–
	/DOS	FILEX	/S
	/EXCLUDE	PIP	/P
	/INTERCHANGE	FILEX	/U
	/LOG	PIP	/W

(Continued on next page)

Table B-1 Monitor Command/System Program Equivalents (Cont.)

Monitor Command	Option	System Utility Program	Option
	/NEWFILES	PIP	/C
	/POSITION:n	PIP	/M
	/QUERY	PIP	/Q
	/NOQUERY	–	–
	/SYSTEM	PIP	/Y
DIBOL		R DIBOL	–
	/ALLOCATE:size	–	[n]
	/ALPHABETIZE	DIBOL	/A
	/CROSSREFERENCE	DIBOL	/C
	/LINENUMBERS	–	–
	/NOLINENUMBERS	DIBOL	/O
	/LIST[:filespec]	–	–
	/OBJECT[:filespec]	–	–
	/NOOBJECT	–	–
	/ONDEBUG	DIBOL	/D
	/WARNINGS	–	–
	/NOWARNINGS	DIBOL	/W
DIFFERENCES		R SRCCOM	–
	/ALLOCATE:size	–	[n]
	/BLANKLINES	SRCCOM	/B
	/COMMENTS	–	–
	/NOCOMMENTS	SRCCOM	/C
	/FORMFEED	SRCCOM	/F
	/MATCH:n	SRCCOM	/L
	/OUTPUT:filespec	–	–
	/PRINTER	–	–
	/SPACES	–	–
	/NOSPACES	SRCCOM	/S
	/TERMINAL	–	–
DIRECTORY		R DIR	–
	/ALLOCATE:size	–	[n]
	/ALPHABETIZE	DIR	/A
	/BADBLOCKS	DUP	/K
	/BEFORE[date]	DIR	/K
	/BEGIN	DIR	/G
	/BLOCKS	DIR	/B
	/BRIEF	DIR, FILEX	/F
	/COLUMNS:n	DIR	/C
	/DATE[date]	DIR	/D
	/DELETED	DIR	/Q
	/DOS	FILEX	/S
	/EXCLUDE	DIR	/P
	/FAST	DIR, FILEX	/F
	/FILES	DUP	/F
	/FREE	DIR	/M
	/FULL	DIR	/E
	/INTERCHANGE	FILEX	/U

(Continued on next page)

Table B-1 Monitor Command/System Program Equivalents (Cont.)

Monitor Command	Option	System Utility Program	Option
	/NEWFILES	DIR	/D
	/OCTAL	DIR	/O
	/ORDER[:category]	DIR	/S
	/OUTPUT:filespec	–	–
	/OWNER:[nnn, nnn]	FILEX	UIC
	/POSITION	DIR	/B
	/PRINTER	–	–
	/REVERSE	DIR	/R
	/SINCE[date]	DIR	/J
	/SORT[:category]	DIR	/S
	/SUMMARY	DIR	/N
	/TERMINAL	–	–
	/TOPS	FILEX	/T
	/VOLUMEID	DUP	/V
DUMP		R DUMP	–
	/ALLOCATE:size	–	[n]
	/ASCII	–	–
	/NOASCII	DUMP	/N
	/BYTES	DUMP	/B
	/END:block	DUMP	/E
	/IGNORE	DUMP	/G
	/ONLY:block	DUMP	/O
	/OUTPUT:filespec	–	–
	/PRINTER	–	–
	/RAD50	DUMP	/X
	/START:block	DUMP	/S
	/TERMINAL	–	–
	/WORDS	DUMP	/W
E		–	–
EDIT		EDIT	EB
	/ALLOCATE:size	–	[n]
	/CREATE	EDIT	EW
	/INSPECT	EDIT	ER
	/OUTPUT:filespec	EDIT	EW
EXECUTE		–	–
	/ALLOCATE:size	–	[n]
	/ALPHABETIZE	DIBOL	/A
	/BOTTOM:n	LINK	/B
	/CODE:type	FORTRAN	/I
	/CROSSREFERENCE[:type[. . . :type]]	DIBOL, MACRO	/C
	/DEBUG[:filespec]	–	–
	/DIAGNOSE	FORTRAN	/B
	/DIBOL	–	–
	/DISABLE:value[. . . :value]	MACRO	/D
	/ENABLE:value[. . . :value]	MACRO	/E
	/EXECUTE[:filespec]	–	–
	/EXTEND	FORTRAN	/E

(Continued on next page)

Table B-1 Monitor Command/System Program Equivalents (Cont.)

Monitor Command	Option	System Utility Program	Option
	/FORTRAN	–	–
	/HEADER	FORTRAN	/O
	/I4	FORTRAN	/T
	/LIBRARY	MACRO	/M
	/LINENUMBERS	–	–
	/NOLINENUMBERS	DIBOL, FORTRAN	/O /S
	/LINKLIBRARY:filespec	–	–
	/LIST[:filespec]	–	–
	/MACRO	–	–
	/MAP[:filespec]	–	–
	/OBJECT[:filespec]	–	–
	/ONDEBUG	DIBOL, FORTRAN	/D
	/OPTIMIZE:type	FORTRAN	/P
	/NOOPTIMIZE:type	FORTRAN	/M
	/PASS:1	MACRO	/P
	/RECORD:length	FORTRAN	/R
	/RUN	–	–
	/NORUN	–	–
	/SHOW[:value]	FORTRAN, MACRO	/L
	/NOSHOW:value	MACRO	/N
	/STATISTICS	FORTRAN	/A
	/SWAP	–	–
	/NOSWAP	FORTRAN	/U
	/UNITS:n	FORTRAN	/N
	/VECTORS	–	–
	/NOVECTORS	FORTRAN	/V
	/WARNINGS	FORTRAN	/W
	/NOWARNINGS	DIBOL	/W
	/WIDE	LINK	/W
FOCAL		R FOCAL	–
FORTRAN		R FORTRAN	–
	/ALLOCATE:size	–	[n]
	/CODE:type	FORTRAN	/I
	/DIAGNOSE	FORTRAN	/B
	/EXTEND	FORTRAN	/E
	/HEADER	FORTRAN	/O
	/I4	FORTRAN	/T
	/LINENUMBERS	–	–
	/NOLINENUMBERS	FORTRAN	/S
	/LIST[:filespec]	–	–
	/OBJECT[:filespec]	–	–
	/NOOBJECT	–	–
	/ONDEBUG	FORTRAN	/D
	/OPTIMIZE:type	FORTRAN	/P
	/NOOPTIMIZE:type	FORTRAN	/M
	/RECORD:length	FORTRAN	/R
	/SHOW[:value]	FORTRAN	/L

(Continued on next page)

Table B-1 Monitor Command/System Program Equivalents (Cont.)

Monitor Command	Option	System Utility Program	Option
	/STATISTICS	FORTRAN	/A
	/SWAP	–	–
	/NOSWAP	FORTRAN	/U
	/UNITS:n	FORTRAN	/N
	/VECTORS	–	–
	/NOVECTORS	FORTRAN	/V
	/WARNINGS	FORTRAN	/W
FRUN		–	–
	/N:n	–	–
	/P	–	–
	/T:n	–	–
GET		–	–
GT OFF		–	–
GT ON		–	–
	/L:n	–	–
	/T:n	–	–
HELP		–	–
	/PRINTER	–	–
	/TERMINAL	–	–
INITIALIZE		–	–
	/BADBLOCKS	DUP	/B
	/DOS	FILEX	/S
	/FILE:filespec	–	–
	/INTERCHANGE	FILEX	/U
	/QUERY	–	–
	/NOQUERY	DUP, FILEX	/Y
	/REPLACE[:RETAIN]	DUP	/R
	/SEGMENTS:n	DUP	/N
	/VOLUMEID[:ONLY]	DUP	/V
INSTALL		–	–
LIBRARY		R LIBR	–
	/ALLOCATE:size	–	[n]
	/CREATE	–	–
	/DELETE	LIBR	/D
	/EXTRACT	LIBR	/E
	/INSERT	–	–
	/LIST[:filespec]	–	–
	/MACRO	LIBR	/M
	/OBJECT[:filespec]	–	–
	/NOOBJECT	–	–
	/PROMPT	LIBR	//
	/REMOVE	LIBR	/G
	/REPLACE	LIBR	/R
	/UPDATE	LIBR	/U
LINK		R LINK	–
	/ALLOCATE:size	–	[n]
	/BOTTOM:n	LINK	/B

(Continued on next page)

Table B-1 Monitor Command/System Program Equivalents (Cont.)

Monitor Command	Option	System Utility Program	Option
	/BOUNDARY:value	LINK	/Y
	/DEBUG[:filespec]	–	–
	/EXECUTE[:filespec]	–	–
	/NOEXECUTE	–	–
	/EXTEND:n	LINK	/E
	/FILL:n	LINK	/Z
	/FOREGROUND[:stacksize]	LINK	/R
	/INCLUDE	LINK	/I
	/LDA	LINK	/L
	/LIBRARY:filespec	–	–
	/LINKLIBRARY:filespec	–	–
	/MAP[:filespec]	–	–
	/PROMPT	LINK	//
	/ROUND:n	LINK	/U
	/RUN	–	–
	/SLOWLY	LINK	/S
	/STACK[:n]	LINK	/M
	/TRANSFER[:n]	LINK	/T
	/WIDE	LINK	/W
LOAD		–	–
MACRO		R MACRO	–
	/ALLOCATE:size	–	[n]
	/CROSSREFERENCE[:type[. . . :type]]	MACRO	/C
	/DISABLE:value[. . . :value]	MACRO	/D
	/ENABLE:value[. . . :value]	MACRO	/E
	/LIBRARY	MACRO	/M
	/LIST[:filespec]	–	–
	/OBJECT[:filespec]	–	–
	/NOOBJECT	–	–
	/PASS:1	MACRO	/P
	/SHOW:value	MACRO	/L
	/NOSHOW:value	MACRO	/N
PRINT		–	–
	/COPIES:n	PIP	/K
	/DELETE	PIP	/D
	/LOG	PIP	/W
	/NOLOG	–	–
	/NEWFILES	PIP	/C
	/QUERY	PIP	/Q
R		–	–
REENTER		–	–
REMOVE		–	–
RENAME		–	–
	/LOG	PIP	/W
	/NOLOG	–	–
	/NEWFILES	PIP	/C
	/QUERY	PIP	/Q

(Continued on next page)

Table B-1 Monitor Command/System Program Equivalents (Cont.)

Monitor Command	Option	System Utility Program	Option
	/REPLACE	–	–
	/NOREPLACE	PIP	/N
	/SETDATE	PIP	/T
	/SYSTEM	PIP	/Y
RESET		–	–
RESUME		–	–
RUN		–	–
SAVE		–	–
SET		–	–
SHOW		–	–
SQUEEZE		–	–
	/OUTPUT:filespec	–	–
	/QUERY	–	–
	/NOQUERY	DUP	/Y
START		–	–
SUSPEND		–	–
TIME		–	–
TYPE		–	–
	/COPIES:n	PIP	/K
	/DELETE	PIP	/D
	/LOG	PIP	/W
	/NOLOG	–	–
	/NEWFILES	PIP	/C
	/QUERY	PIP	/Q
UNLOAD		–	–

INDEX

INDEX (Cont.)

INDEX (Cont.)

INDEX (Cont.)

INDEX (Cont.)

INDEX (Cont.)

INDEX (Cont.)

INDEX (Cont.)

INDEX (Cont.)

INDEX (Cont.)

INDEX (Cont.)

INDEX (Cont.)

INDEX (Cont.)

INDEX (Cont.)

INDEX (Cont.)

INDEX (Cont.)

INDEX (Cont.)

INDEX (Cont.)

READER'S COMMENTS

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

Did you find errors in this manual? If so, specify by page.

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

Is there sufficient documentation on associated system programs required for use of the software described in this manual? If not, what material is missing and where should it be placed?

Please cut along this line.

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

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Non-programmer interested in computer concepts and capabilities

Name _____ Date _____

Organization _____

Street _____

City _____ State _____ Zip Code or Country _____

---------- Fold Here ----------

---------- Do Not Tear - Fold Here and Staple ----------

FIRST CLASS
PERMIT NO. 33
MAYNARD, MASS.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

Postage will be paid by:

digital

Software Documentation
146 Main Street ML 5-5/E39
Maynard, Massachusetts 01754

digital

digital equipment corporation

Printed in U.S.A.