# RT-11
# System Reference Manual

Order No. DEC-11 -ORUGA-C-D, DN1, DN2

digital equipment corporation . maynard. massachusetts

First Printing, September 1973
Revised: October 1974
June 1975
July 1975
January 1976

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

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

Digital Equipment Corporation assumes no responsibility for the use or reliability of its Software on equipment that is not supplied by DIGITAL.



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

The following are trademarks of Digital Equipment Corporation:

| | | |
|---|---|---|
| DIGITAL | **DECsystem-10** | MASSBUS |
| DEC | **DECtape** | OMNIBUS |
| PDP | DIBOL | *OS/8* |
| DECUS | EDUSYSTEM | PHA |
| UNIBUS | FLIP CHIP | RSTS |
| COMPUTER LABS | **FOCAL** | RSX |
| COMTEX | INDAC | **TYPESET-8** |
| DDT | LAB-8 | **TYPESET-10** |
| DECCOMM | **DECsystem-20** | **TYPESET-11** |

CONTENTS

TABLES



J

## FIGURES

# PREFACE

This manual describes the use of the RT-11 Operating System. It assumes the reader is familiar with Computer Software fundamentals and has had some exposure to assembly language programs. The section "Additional and Reference Material" later in this Preface lists documents that may prove helpful in reviewing those areas. The Glossary provides definitions of technical terms used in the manual.

The user who is unfamiliar with RT-11 should first read those chapters of interest (see "Chapter Summary" below) to become familiar with system conventions. Having gained familiarity with RT-11, the user can then reread the manual for specific information.

## Chapter Summary

Chapter 1 discusses system hardware and Software requirements. It describes general system operations and lists specific components available under RT-11.

Chapter 2 introduces the user to system conventions and monitor/memory layout. It describes in detail the keyboard commands for controlling jobs and implementing user programs.

Chapters 3 through 8 describe the system utility programs EDIT, PIP, MACRO, LINK, LIBR, and ODT, respectively. These programs (a text editor, file transfer program, assembler, linker, librarian, and debugging program) aid the user in creating text files and producing assembly-language programs.

Chapter 9, which describes programmed requests, is of particular interest to the experienced programmer. It describes call sequences that allow the user to access system monitor Services from within assembly-language programs.

Chapters 10 and 11 describe the 8K Assembler and EXPAND programs, respectively. These programs are useful in RT-11 installations with minimum memory configurations.

Chapter 12 describes the BATCH command language for RT-11. In BATCH mode, the RT-11 system can be left to run unattended for long periods of time.

The appendixes summarize the contents of the manual and describe additional system utility programs that can be used for extended system operations. These programs include SRCCOM (a source file comparison program); FILEX (a file translation program that allows

transfer of files between RT-11 and other DIGITAL operating Systems); PATCH and PATCHO (patching programs); DUMP (a file dump program); and SYSLIB (a library of programmed requests for FORTRAN users).

## Version History

The current RT-11 system (monitor) is Version 2C (V2C). Each system component (monitors and Utilities) is assigned a Software identification number in the form Vxx-xx. Current identification numbers for V2C are listed in the RT-11 System Release Notes (DEC-11-ORNRA-A-D). To determine whether the correct Version of a component is in use, examine its identification number and compare it with the list. (The procedure for examining the Version number varies. Most system programs provide a special command; others print the version number when an output listing is requested. Consult the approiate chapter or appendix of this manual for each component.)

NOTE

Throughout this manual, any references to V2 or V2B of RT-11 will pertain also to V2C. The RT-11 System Reiease Notes contain a comprehensive list of differences between V2C and previous Versions of RT-11 (V2B, V2, V1).

Change bars and asterisks in the outermost margins of the manual are used to denote changes made to the text since the Version 2 release (DEC-11-ORUGA-B-D). The date July 1975 in the lower outside corner of a page indicates that the page was changed as a result of a release-independent update that occurred in July, 1975. The date January 1976 in the lower outside corner of the page indicates that the page was changed specifically as a result of the V2C update.

The user who is already familiar with the Version 2B RT-11 System Reference Manual (DEC-11-ORUGA-C-D,DN1) should first read the RT-11 System Release Notes document to note the major differences between V2B and V2C, and then read those pages of the RT-11 System Reference Manual that have changed as a result of the V2C update (identified by the date January 1976). The RT-11 System Generation Manual (DEC-11-ORGMA-A-D) should also be read if customization for special devices and features is required.

The user who is familiar with only the Version 2 RT-11 System Reference Manual (DEC-11-ORUGA-B-D) should read the following in addition to those items mentioned in the preceding Paragraph:

| | | |
|---|---|---|
| Chapter 2 | (System Communication) | - Tables 2-2, 2-3, and 2-5 |
| Chapter 3 | (Text Editor) | Section 3.6.5.6 |
| Chapter 9 | (Programmed Requests) | - Sections 9.1 and 9.1.3.6 |
| Chapter 12 | (BATCH) | Entire Chapter |
| Appendix H | (F/B Programming And Device Handlers) | - Sections H.2.4 and H.2.5 |
| Appendix 0 | (SYSLIB) | Entire Appendix |

Finally, the user familiar with only the Version 1 RT-11 System Reference Manual (DEC-11-ORUGA-A-D) should read this entire manual with these exceptions:

Chapter 3 (Text Editor) - note Section 3.7
Chapter 5 (MACRO Assembler) - note Section 5.7
Chapter 8 (ODT) - note restrictions in Section 8.1
Chapter 10 (EXPAND)
Chapter 11 (ASEMBL)
Appendix L (PATCH)

While knowledge of Versions 2 and 2B is sufficient for use of V2C, knowledge of Version 1 is not; the user with Version 1 knowledge only should carefully read the manual.

## Additional and Reference Material

The following manuals provide an introduction to the PDP-11 Computer family and the basic PDP-11 instruction set:

PDP-11 Paper Tape Software Programming Handbook** (DEC-11-XPTSA-B-D)
PDP-11 Processor Handbook*
PDP-11 Peripherals Handbook*

The following manual provides an introduction to the use of RT-11 by presenting a simple demonstration of basic operating procedures:

RT-11 System Generation Manual* (DEC-11-ORGMA-A-D)

These manuals describe the capabilities of the optional high-level language components:

BASIC/RT-11 Language Reference Manual** (DEC-11-LBACA-D-D)
PDP-11 FORTRAN Language Reference Manual** (DEC-11-LFLRA-B-D)
RT-11/RSTS/E FORTRAN IV User's Guide** (DEC-11-LRRUA-A-D)

Summaries of the features provided by each language appear in this manual in Appendixes F and G respectively.

Two PDP-11 system manuals are helpful when using FILEX (Appendix J) to convert programs between DOS, RSTS, and RT-11 formats:

PDP-11 Resource Sharing Time-sharing System User's Guide** (DEC-11-ORSUA-D-D)
DOS/BATCH Handbook** (DEC-11-ODBHA-A-D)

Users of display hardware may wish to refer to the appropriate hardware manual:

GT40/42 User's Guide*** (398150)
GT44 User's Guide*** (398250)
VT11 Graphic Display Processor Manual*** (79H650)
DECscope User's Manual*** (EK-VT50-OP)

The experienced programmer will want to read the following manual:

RT-11 Software Support Manual* (DEC-11-ORPGA-B-D)

---

*Included in the RT-11 Software Kit
**May be ordered from the DIGITAL Software Distribution Center
***May be ordered from DIGITAL Communication Services

Consult the following for a list of all manuals available in the RT-11 Software documentation set:

RT-11 Documentation Directory* (DEC-ll-ORDDA-A-D)

## Documentation Conventions

Conventions used throughout this manual include the following:

1. Actual Computer output is used in examples wherever possible. When necessary, Computer output is underlined to differentiate from user responses.

2. A line feed (Character or key) is represented in the text as <LF>; a carriage return (Character or key) is represented as <CR>. Unless otherwise indicated, all commands and command strings are terminated by a carriage return.

3. Terminal, console terminal, and teleprinter are general terms used throughout all RT-11 documentation to represent any terminal device, including DECwriters, displays, and Teletypes****. RP02 is a generic term used to represent both the RPllC/RP02 and RPllE/RPR02 disks.

4. Several characters in system commands are produced by typing a combination of keys concurrently; for example, the CTRL key is held down while typing an O to produce a command which causes suppression of teleprinter output. Key combinations such as this are documented as CTRL 0, CTRL C, SHIFT N, and so forth.

*Included in the RT-11 Software Kit
****Teletype is a registered trademark of the Teletype Corporation.

# CHAPTER 1

# RT-11 OVERVIEW

RT-11 is a single-user programming and operating system designed for the PDP-11 series of computers. This system permits the use of a wide range of peripherals and up to 28K of either solid state or core memory (hereafter referred to as memory).

RT-11 provides two operating environments: Single-Job Operation, and a powerful Foreground/Background (F/B) capability(1).

Single-Job Operation allows only one program to reside in memory at any time; execution of the program continues until either it is completed or it is physically interrupted by the user at the console.

In a Foreground/Background environment, two independent programs may reside in memory. The foreground program is given priority and executes until it relinquishes control to the background program; the background program is allowed to execute until control is again required by the foreground program, and so on. This sharing of system resources greatly increases the efficiency of processor usage.

To handle both operating environments, RT-11 offers two completely compatible and versatile monitors (Single-job and F/B); either monitor provides complete user control of the system from the console terminal keyboard. Monitor commands which allow the user to direct Single-job, foreground, and background operations are described in Chapter 2.

In addition to the monitor facilities, RT-11 offers a full complement of system programs; these allow program development using high level languages such as FORTRAN IV and BASIC or assembly language (MACRO or EXPAND/ASEMBL). System programs are summarized in Section 1.2 and are discussed in detail in individual chapters and appendixes of this manual.

---

(1) The uses and advantages of each environment are outlined later in this chapter.

## 1.1 PROGRAM DEXELOPMENT

Computer Systems such as RT-11 are often used extensively for program development. The programmer makea use of the programming "tools" available on his system to develop programs which will perform functions specific to his needs. The number and type of "tools" available on any given System depend on a good many factors--the size of the System, its application and its cost, to name a few. Most DIGITAL Systems, however, provide several basic program development aids: these generally fnclude an editor, assembler, linker, debugger, and often a librarian; a high level language (such as FORTRAN IV or BASIC) is also usually available.

An editor is used to create and modify textual material. Text may be the lines of code which make up a Source program written in some programming language, or it may be data; text may be reports, or memos, or in fact may consist of any subject matter the user wishes. In this respect using an editor is analogous to using a typewriter--the user sits at a keyboard and types text. But the advantages of an editor far exceed those of a typewriter because once text has been created, it can be modified, relocated, replaced, merged, or deleted-all by means of simple editing commands. When the user is satisfied with his text, he can save it *on a* storage device where it is available for later reference.

ff the editor is used for the purpose of writing 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), an intermediary program is necessary which will convert source code into the instructions the Computer can execute. This is the function of an assembler.

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

The output produced by the assembler is called object output because it is composed of object (or binary) code. On PDP-11 Systems, the object output is called a module and contains the user's source program in the binary language which 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 in conjunction with other programs (or modules) in order 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. It is the purpose of linker to perform this relocation.

The linker combines and relocates separately assembled object programs. The output produced by the linker consists of a load module, which ie the final linked program ready for execution. The user can, at his Option, request a load map which displays all addresses assigned by the linker.

Very rarely is a program created which does not contain at least one unintentional error, either in the logic of the program or in its coding. Errors may be discovered by the programmer while he is editing his program, or the assembler may find errors during the assembly process and inform the programmer by means of error codes. The linker may also catch certain errors and issue appropriate messages. Often, however, it is not until execution that the user discovers his 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 the programmer in determining the cause of his error.

A debugging program allows the user to interactively control the execution of his program. With it, he 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 the need of re-editing and re-assembling.

When programs are successfully written and executed, they may be useful to other programmers. Often routines which are common to many programs (such as I/O routines) or sections of code which 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 the programmer's work by providing an alternate means of writing a source program other than assembly language mnemonics. Generally, high level languages are easy to learn--a single command may cause the Computer to perform many machine language instructions. The user does 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 which allows the user to solve equations and formulas as though he were using a calculator. Assembling and linking are done automatically so that the user can concentrate on solving the problem rather than 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 the user of RT-11.

## 1.2 SYSTEM SOFTWARE COMPONENTS

The following is a brief summary of the RT-11 system programs:

1. The Text Editor (EDIT, described in Chapter 3) is used to create or modify source files for use as input to language processing programs such as the assembler or FORTRAN. EDIT contains powerful text manipulation commands for quick and easy editing of a text file. EDIT also allows use of a VT11 display processor (such as the GT44), if one is part of the hardware configuration (see Section 1.3).

2. The MACRO Assembler (Chapter 5) brings the capabilities of macros to the RT-11 system with 12K (or more) memory. (Macros are instructions in a source or command language which are equivalent to a specified sequence of machine

instructions or commands.) The assembler accepts source files written in the MACRO language and generates a relocatable object module to be processed by the Linker before loading and execution. Cross reference listings of assembled programs may be produced using CREF in conjunction with the MACRO Assembler.

3. EXPAND (Chapter 10) is used in an 8K F/B job area or 8K Systems (or in larger Systems with programs of great size) to expand macros in an assembly language program into macro-free source code, thus allowing the program to be assembled in 8K using ASEMBL.

4. ASEMBL (Chapter 11) is an assembler designed for use in an 8K RT-11 system, an 8K F/B job area, or larger systems where symbol table space is a factor. ASEMBL is a subset of MACRO-11 with more limited features. (CREF is not available under ASEMBL.)

5. The Linker (LINK, described in Chapter 6) fixes (i.e., makes absolute) the values of relocatable symbols and converts the relocatable object modules of compiled or assembled programs and subroutines into a load module which can be loaded and executed by RT-11. LINK can automatically search library files for specified modules and entry points; it can produce a load map (which lists the assigned absolute addresses) and can provide automatic overlay capabilities to very large programs. The Linker can also produce files suitable for running in the foreground.

6. The Librarian (LIBR, see Chapter 7) allows the user to create and maintain his own library of functions and routines. These routines are stored on a random access device as library files, where they can be referenced by the Linker.

7. The Peripheral Interchange Program (PIP, see Chapter 4) is the RT-11 file maintenance and Utility program. It is used to transfer files between all devices which are part of the RT-11 System, to rename or delete files, and to obtain directory listings.

8. SRCCOM (Source Compare, described in Appendix K) allows the user to perform a Character-by-Character comparison of two or more text files. Differences can be listed in an output file or directly on the line printer or terminal, thus providing a fast method of determining, for example, if all edits to a file have been correctly made.

9. FILEX (Appendix J) allows file transfers to occur between DECtapes used under the DECsystem-10 or PDP-11 RSTS system, and DECtape and disk used under the DOS/BATCH System, and any RT-11 device.

10. The PATCH Utility program (Appendix L) is used to make minor modifications to memory image files (output files produced by the Linker); it is used on files which do or do not have overlays. PATCHO (Appendix M) is used to make minor modifications to files in object format (output files produced by the FORTRAN Compiler and the Librarian, or MACRO and ASEMBL assemblers).

11. ODT (On-line Debugging Technique, described in Chapter 8) aids in debugging assembled and linked object programs. It can print the contents of specified locations, execute all or part of the object program, single step through the object program, and search the object program for bit Patterns.

12. DUMP (Appendix 1) is used to print for examination all or any part of a file in octal words, octal bytes, ASCII and/or RAD50 characters (see Chapter 5).

13. BATCH (Chapter 12) is a complete job control language that allows RT-11 to operate unattended. The BATCH stream may be composed of RT-11 monitor commands or system-independent BATCH jobs (jobs that will run on any DIGITAL system supporting the BATCH Standard; currently RT-11 and RSX-11D). BATCH streams can be executed under the Single-Job Monitor or in the background under the F/B Monitor.

14. The RT-11 FORTRAN System Subroutine Library (SYSLIB, Appendix 0) is a collection of FORTRAN callable routines that make the programmed requests and various Utility functions available to the FORTRAR programmer. SYSLIB also provides a complete string manipulation package and two-word integer package for RT-11 FORTRAN.

BASIC and FORTRAR IV are two high level languages available under RT-11. Summaries of their language features and commands are provided in Appendixes F and G of this manual.

## 1.3 SYSTEM HARDWARE COMPONENTS

The minimum RT-11 system (that is, one that does not use the F/B capability) requires a PDP-11 series Computer with at least 8K of memory, a random-access device, and a console terminal. The F/B capability requires at least 16K of memory and a line frequency clock. For specific hardware/software interdependent requirements, refer to the RT-11 System Release Notes.

Devices supported by RT-11 are listed in Table l-l. The third (middle) column lists devices for which support is initially provided in the system as distributed; these devices can be used with no modification (to either the monitor tables or the handlers) necessary. The cevices in the fourth column are supported after simple modifications to the monitor tables or handlers. The system customization section of the RT-11 System Generation Manual describes how to make these modifications. The fifth column lists devices for which no support is provided, 'but which may be intcrfaced by the user. Currently, the RS64 disk is the only device in this category, and instructions for its interface are provided in the RT-11 Software Support Manual.

Consult the RT-11 System Generation Manual for modifications that may be made to existing system devices (for example, varying the baud rate of a terminal).

**Table 1-1**
RT-11 Hardware Components

| Category | Controller | System-Installed Devices | Devices Requiring System Modification | User-Installed Devices |
|---|---|---|---|---|
| DISK | | | | |
| DECpack Cartridge | RK11 | RK05 | | |
| Fixed-head | RF11<br>RC11 | RS11 | | RS64 |
| | RH11 | RJS03 | RJS04 | |
| Removable Pack | RP11 | RP02 | RP03 | |
| Diskette | RX11 | RX01 | RX01 (second Controller) | |
| DECTAPE | TC11 | TU56 | | |
| MAGTAPE | TM11/TMA11 | TU10,TS03 | | |
| | RH11 | TJU16 | | |
| CASSETTE | TA11 | TU60 | | |
| HIGH-SPEED PAPERTAPE READER/PUNCH | PC11<br>PR11 | PC11 (both)<br>PR11 (reader only) | | |
| LINE PRINTER | LS11 | LS11, LA180 | | |
| | LV11 | LV11 (printer only) | | |
| | LP11 | all LP11 controlled printers | | |
| CARD READER | CR11 | | CR11 | |
| | CM11 | | CM11 | |
| TERMINAL | DL11 | LT33, LT35 | | |
| | | LA30P, LA36, VT50, VT52, VT05 | LA30S | |
| DISPLAY PROCESSOR | VT11 | VR14-L,VR17-L | | |
| CLOCK | | KW11-L | | |

RT-11 operates in environments ranging from 8K to 28K words of memory. Reconfiguration for different memory sizes is unnecessary--the same system device operates on any PDP-11 processor with 8K to 28K of memory and makes use of all memory available.

## 1.4 USING THE RT-11 SYSTEM

As mentioned earlier in the chapter, the RT-11 system offers two complete operating environments. Each is controlled by a single user from the console terminal keyboard by means of an appropriate monitor--Single-Job or Foreground/Background. Both monitors are completely compatible and allow full user interaction with all features which are a part of the operating environment in use.

The choice of which environment to use, and, consequently, which monitor to run, depends upon the needs of the user. The next two sections provide information useful in determining which monitor is more suitable for certain applications.

### 1.4.1 RT-11 Single-Job Monitor

The RT-11 Single-Job Monitor provides a Single-user, Single-program system which can operate in as little as 8K of memory. Since the Single-Job Monitor itself requires approximately one-half the memory space needed by the Foreground/Background Monitor, this system is ideal for extensive program development work; a much larger area of memory is available for the user program and its buffers and tables. Programs requiring extremely high data rates are best run in the Single-Job environment, since interrupts can be serviced at a much higher rate.

All system programs' (listed in Section 1.2) can be used under the Single-Job Monitor, and many of the features of the Foreground/Background Monitor (i.e., KMON commands and programmed requests not used to control foreground jobs) are supported.

In effect, the Single-Job Monitor is much smaller and slightly faster than the Foreground/Background Monitor; it can best be used when program size is the important factor.

### 1.4.2 RT-11 Foreground/Background Monitor

Quite often the central processor of a Computer system may spend a large percentage of time waiting for some external event to occur, the most common event being the completion of an I/O transfer (this is particularly true of real time jobs). Many users would like to take advantage of this unused capacity to accomplish other lower-priority tasks such as further program development or complex data analysis. The Foreground/Background system provides this capability.

In a Foreground/Background system the foreground job is the time-critical, on-line job, and is given top priority; whenever possible the processor runs the foreground job. Bowever, when the foreground job reaches a state in which no more processinq can be done

until some external event occurs, the monitor will try to run the lower priority background job. The background job then runs until the foreground job is again in a runnable state, at which point the processor will interrupt the background job and resume the foreground job.

In general, the RT-11 Foreground/Background System is designed to allow a time-critical job to run in the foreground, while the background does non-time-critical jobs, such as program development. (All RT-11 system programs run as the background job in a F/B System.1 Thus, the user can run FORTRAR, BASIC, MACRO, etc. in the background while the foreground may be collecting data and storing and/or analyzing it.

Most user programs written for an RT-11 System can be linked (using the Linker described in Chapter 6) to run as the foreground job. There are a few coding restrictions, and these are explained in Appendix H, F/B Programming and Device Handlers. A foreground program has access to all of the features available to the background job (opening and closing files, reading and writing data, etc.). In addition, the F/B System gives the user the ability to set timer routines, suspend and resume F/B jobs, and send data and messages between the two jobs.

### 1.4.3 Facilities Available Only in RT-11 F/B

As mentioned previously, RT-11 F/B allows the user to write and execute two independent programs. Some features which are available only to the F/B user includer

1. Mark Time--This facility allows user programs to set clock timers to run for specified amounts of time. When the timer runs out, a routine specified by the user is entered. There may be as many mark time requests as desired, providing system queue space is reserved (see .QSET, Chapter 9).

2. Timed Wait--This feature allows the *user* program to "sleep" until the specified time increment elapses. Typically, a program may need to sample data every few seconds or even minutes. While the program is idle, the other job can *run.* The timed wait accomplishes this; when the time has elapsed, the issuing job is again runnable (see .TWAIT, Chapter 9).

3.. Send Data/Receive Data--It is possible, under RT-11 F/B, to have the foreground and background programs communicate with one another. This is accomplished with the send/receive data functions. Using this facility, one program sends messages (or data) in variable size blocks to the other job. This can be used, for example, to pass data from a foreground collection program directly to a background analysis program (see .SDAT/.RCVD, Chapter 9).

# CHAPTER 2

# SYSTEM COMMUNICATION

The monitor is the hub 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.*

The user communicates with the monitor through programmed requests and keyboard ccmmands. The keyboard commands (described in Section 2.7) are used to load and run programs, start or restart programs at specific addresses, modify the contents of memory, and assign and deassign alternate device names.

Programmed requests (described in detail in Chapter 9) are source program instructions which pass arguments to the monitor and request monitor Services. These instructions allow user assembly language programs to utilize the available monitor features.

## 2.1 START PRCCEDURE

After the system has been built (see the RT-11 System Generation Manual), the monitor can be loaded into memory from disk or DECtape as follows:

1. Press HALT.
2. Mount the system device on unit 0 (or the appropriate unit if a unit other than 0 is to be used).
3. WRITE PROTECT the System unit.

If the hardware configuration includes a hardware bootstrap capable of booting the system device,

1. Set the switch register to the appropriate address and press LOAD ADRS.
2. If a second address is required, set the switch register to that address.
3. Press START.

If a hardware bootstrap is not available, or if an RK disk unit other than 0 is to be used as the system device, one of the following bootstraps must be entered manually using the Switch Register. First set the Switch Register to 1000 and press the LOAD ADES switch. Then set the Switch Register to the first value shown for the appropriate bootstrap and raise the DEPosit switch. Continue depositing the values shown.

| DECtape (RK11,RK05) | (RK Disk other than Unit 0) | Disk (RF11) | (RJS03/4) | (RP11/RP02) | (RX11/RX01) |
|---|---|---|---|---|---|
| 12700 | 12700 | 12700 | 12700 | 12705 | 12705 | 12702 |
| 177344 | 177406 | 177406 | 177466 | 172044 | 176716 | 1002n7** |
| 12710 | 12710 | 12760 | 5010 | 12745 | 12715 | 12701 |
| 177400 | 177400 | xxxxxx * | 5040 | 177400 | 177400 | 177170 |
| 12740 | 12740 | 4 | 12740 | 12745 | 12745 | 130211 |
| 4002 | 5 | 12700 | 177400 | 71 | 5 | 1776 |
| 5710 | 105710 | 177406 | 12740 | 32715 | 105715 | 112703 |
| 100376 | 100376 | 12710 | | 100200 | 100376 | 7 |
| 12710 | 5007 | 177400 | 105710 | 1775 | 5007 | 10100 |
| 3 | | 12740 | 100376 | 100762 | | 10220 |
| 105710 | | 5 | 5007 | 5007 | | 402 |
| 100376 | | 105710 | | | | 12710 |
| 12710 | | 100376 | | | | 1 |
| | | 5007 | | | | 6203 |
| 105710 | | | | | | 103402 |
| 100007 | | | | | | 112711 |
| | | | | | | 111023 |
| | | | | | | 30211 |
| | | | | | | 1776 |
| | | | | | | 100756 |
| | | | | | | 103766 |
| | | | | | | 105711 |
| | | | | | | 100771 |
| | | | | | | 5000 |
| | | | | | | 22710 |
| | | | | | | 240 |
| | | | | | | 1347 |
| | | | | | | 122702 |
| | | | | | | 247 |
| | | | | | | 5500 |
| | | | | | | 5007 |

* xxxxxx = 20000 for unit 1
40000 for unit 2
60000 for unit 3
100000 for unit 4
120000 for unit 5
140000 for unit 6
160000 for unit 7

** n = 4 for unit 0
6 for unit 1

When all the values have been entered, set the switches to 1000 and press the LOAD ADES and START switches.

The monitor loads into memory and prints one of the following identification messages followed by a dot (.) on the terminal:

```
RT-11SJ V02C-xx
RT-11FB V02C-xx
```

The message printed indicates which monitor (Single-Job or F/B) has been loaded; the user may determine which is to be loaded during the system build Operation.

After the message has printed, the system device should be WETTE ENABLED. The monitor is ready to accept keyboard commands.

To bring up an alternate monitor while under control of the one currently running (in this case, F/B), run PIP to perform the following operations:

1. Preserve the running monitor by renaming it to yyyyyy.SYS (the actual name yyyyyy is not significant, although it is suggested that yyMNSJ for Single-Job and yyMNFB for Foreground/Background be used to be consistent with system conventions; yy in this case represents the disk type):

```
.R PIP
*RK0:RKKNFB.SYS=RK0:MONITR.SYS/R/Y
?REBOOT?
```

2. Rename the desired monitor to MONITR.SYS:

```
*RK0:MONITR.SYS=RK0:RKMNSJ.SYS/R/Y
?REBOOT?
```

3. Write the new bootstrap from the new MONITR.SYS file (using the PIP /U Option; A is a dummy filename, which must be present in the command line):

```
*RK0:A=RK0:MONITR.SYS/U
```

4. Reboot the system.

```
*RK0:/O

RT-11SJ    V02C-02

.
```

Refer to the RT-11 System Generation Manual for an example of switching monitors.

This page intentionally blank.

## 2.2 SYSTEM CONVENTIONS

Special Character commands, file naming procedures and other conventions that are standard for the RT-11 system are described in this section. The user should be familiar with these conventions before running the System.

### 2.2.1 Data Formats

The RT-11 system makes use of five types of data formats: ASCII, object, memory image, relocatable image, and load image.

File8 in ASCII format conform to the American National Standard Code for Information Interchange, in which each Character is represented by a I-bit code. File8 in ASCII format include program source files created by the Editor, listing and map files created by various system programs, and data files consisting of alphanumeric characters. A chart containing ASCII character codes appears in Appendix C.

File8 in object format consist of data and PDP-11 machine language code. Object files are those output by the assembler or FORTRAN Compiler and are used as input to the Linker.

The Linker can output files in memory image format (.SAV), relocatable Image format (.REL), or load image format (.LDA).

A memory image file (.SAV) is a 'picture' of what memory will look like when a program is loaded. The file itself requires the same number of disk blocks as the corresponding number of 256-word memory blocks.

A relocatable image file (.REL) is one which can be run in the f oreground. It differs from a memory image file in that the file is linked as though its bottom address were 0. When the program is called (using the monitor FRUN command), the file is relocated as it is loaded into memory. (A memory image file requires no such relocation.)

A load fmage (or .LDA) file may be produced for compatibility with the PDP-11 Paper Tape System and is loaded by the absolute binary loader. LDA files can be loaded and executed in stand-alone environments without relocation.

### 2.2.2 Prompting Character8

The following table summarizes the characters typed by ET-11 to indicate to the user either that the system is awaiting user response or to specify which job (foreground or background) is producing output:

Table 2-l
Prompting Character8

| Character | Meaning |
|---|---|
| . | The Keyboard Monitor is waiting for a command (see Section 2.3.2). |
| * | The Command String Interpreter is waiting for a command string specification as explained in Sections 2.3.3 and 2.5. |
| ↑ | When the console terminal is being used as an Input file, the uparrow prompt8 the user to enter information from the keyboard. If the input is entered under EDIT or BASIC (or any program that accepts input in special terminal mode as described in Chapter 9), the characters entered are not echoed. Typing a CTRL Z marks the end-of-file. |
| > | The > Character is used (under the F/B Monitor and only if a foreground job is active) to identify which job, foreground or background, is producing the output currently appearing on the console terminal. Each time output from the background job is to appear, B> is printed first, followed by the output. If the foreground job ie to print output, F> is typed first. B> and F> are also printed as a result of the CTEL B and CTRL F commands described in Table 2-4. |

### 2.2.3 Physical Device Names

Devices are referenced by means of a Standard two-Character device name. Table 2-2 lists each name and its related device. If *no* unit number is specified for devices which have more than one unit, unit 0 is assumed.

Table 2-2
Permanent Device Names

| Permanent Name | I/O Device |
|---|---|
| CR: | Card Reader (CR11/CM11). |
| CTn: | TA11 cassette (n is the unit number, 0 or 1). |
| DK: | The default logical storage device for all files. DK is initially the same as SY: (see below), but the assignment (as a logical device name) can be changed with the ASSIGN Command (Section 2.7.2.4). |
| DKn: | The specified unit of the same device type as DK. |
| DPn: | RP02 disk (n is an integer in the range 0-7). |
| DSn: | RJS03/4 fixed-head disks (n is in the range 0-7). |
| DTn: | DECtape n, where n is a unit number (an integer in the range 0 to 7, inclusive). |
| DXn: | RX01 Floppy disk (n is 0 or 1). |
| LP: | Line printer. |
| MMn: | TJU16 magtape (n is in the range 0-7). |
| MTn: | TM11 (industry compatible) magtape (n is an integer between 0 and 7, inclusive). |
| PP: | High-Speed paper tape punch. |
| PR: | High-Speed paper tape reader. |
| RF: | RF11 fixed-head disk drive. |
| RKn: | RK disk cartridge drive n (n is in the range 0 to 7 inclusive). |
| SY: | System device; the device and unit from which the system is bootstrapped. (RT-11 allows bootstrapping from any RK unit; refer to Section 2.1.) The assignment as a logical device name can be changed with the ASSIGN command (Section 2.7.2.4). |
| SYn: | The specified unit of the same device type as that from which the system was bootstrapped. |
| TT: | Terminal keyboard and printer. |

In addition to the fixed names shown in Table 2-2, devices can be assigned logical names. A logical name takes precedence over a physical name and thus provides device independence. With this feature a program that is coded to use a specific device does not need to be rewritten if the device is unavailable. Refer to Section 2.7.2.4 for instructions on assigning logical names to devices.

### 2.2.4 File Names and Extensions

Files are referenced symbolically by a name of one to six alphanumeric characters followed, optionally, by a period and an extension of up to three alphanumeric characters. (Excess characters in a filename may cause an error message.) The extension to a filename generally indicates the fonnat of a file. It is a good practice to conform to

the standard filename extensions for RT-11. ff an extension is not specified for an input.or output file, most System programe assign appropriate default extenaions. Table 2-3 liets the Standard extensions used in RT-11.

Table 2-3
File Name Extension8

| Extension | Meaning |
|---|---|
| .BAD | File6 with bad (unreadable) blocks; this extension can be assigned by the user whenever bad areas occur on a device. The .BAD extension 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. |
| .CTL | BATCH control file generated by the BATCH Compiler. |
| . CTT | BATCH internal temporary file. |
| .DAT | BASIC or FORTRAN data file, |
| .DIR | Directoryliating file |
| .DMP | DUMP output file. |
| .FOR | FORTRAR IV source file (FORTRAR input). |
| .LDA | Absolute binary file (optional Linker output). |
| .LLD | Library listing file. |
| .LOG | BATCB log file. |
| .LST | Listing file (MACRO or FORTRAN output). |
| .MAC | MACRO or EXPAND source file (MACRO, EXPAND, SRCCOM input). |
| .MAP | Map file (Linker output). |
| .OBJ | Relocatable binary file (MACRO, ASEMBL, FORTRAR IV output, Linker input, LIBR input and output). |
| .PAL | Output file of EXPAND (the MACRO expander program), input file of ASEMBL. |
| .REL | Foreground job relocatable image (Linker output, default for monitor FRUN command). |
| .SAV | Memory image or SAVE file; default for R, RUN, SAVE and GET Keyboard Monitor commands; also default for output of Linker. |
| . SOU | Temporary source file generated by BATCH. |
| .SYS | System files and handlers. |

If a filename with a blank extension is to be used in a command line In which a default extension is assumed (by either the monitor or a system program), the user must insert a period after the filename to indicate that there is no extension. For example, to run the file TEST, type:

```
.RUN TEST.
```

If the period after the filename is not glven, the monitor assumes the .SAV extension and attempts to run a file named TEST.SAV.

### 2.2.5 Device Structures

RT-11 devices are categorized by the physical structure of the device and the way in which the device allows information to be processed.

All RT-11 devices are either random-access or sequential-access devices Random-access devices allow blocks of data to be processed in a random order -- that is, independent of the data's physical location on the device or its location relative to any other information. All disks and DECtape fall into this category. Random-access devices are sometimes also called block-replaceable devices, because individual data blocks can be manipulated (rewritten) without affecting other data blocks on the device. Sequential-access devices require that data be processed sequentially; the order of processing data must be the same as the physical order of the data. RT-11 devices that are considered sequential devices are magtape, cassette, paper tape, card reader, line printer, and terminal.

File-structured devices are those devices that allow the storage of data under assigned filenames. RT-11 devices that are file-structured include all disks, DECtape, magtape, and cassette. Nonfile-structured devices, on the other hand, are those used to contain a single logical collection of data. These devices are used generally for reading and listing information, and include line printer, card reader, terminal, and paper tape devices.

Finally, file-structured devices are classified further as RT-11 directory-structured devices if they provide a Standard RT-11 directory at the beginning of the device (the Standard RT-11 directory is defined in the RT-11 Software Support Manual). The directory contains information about all files stored on the device and is updated each time a file is moved, added, or deleted from the device. RT-11 directory-structured devices include all disks and DECtapes. NonRT-11 directory-structured devices are file-structured devices that do not have the Standard RT-11 directory structure at their beginning. For example, some devices, such as magtape and cassette, have directory-type information stored at the beginning of each file; the device must be read sequentially to obtain all information about all files.

It is possible to interface a device to the RT-11 system with a user-defined directory structure; procedures are explained in the RT-11 Software Support Manual.

## 2.3 MONITOR SOFTWARE COMPONRNTS

The main RT-11 monitor Software components are:

Resident Monitor (RMON)

Keyboard Monitor (KMON)

User Service Routine (USR) and Command String Interpreter (CSI)

Device Handlers

The reader may find Figure 2-l helpful while reading the following descriptions.

### 2.3.1 Resident Monitor (RMON)

The Resident Monitor is the only pennanently memory-resident part of RT-11. The programmed requests for all Services of RT-11 are handled by RMON. RMON also contains the console terminal Service, error processor, system device handler, EMT processor, and system tables.

### 2.3.2 Keybonrd Monitor (KMON)

The Keyboard Monitor provides communication between the user at the console and the RT-11 system. Monitor commands allow the user to assign logical names to devices, run programs, load device handlers, and control F/B operations. A dot at the left margin of the conaole terminal page indicates that the Keyboard Monitor is in memory and ie waiting for a user command.

### 2.3.3 User Service Routine (USR)

The User Service Routine provides support for the RT-11 file structure. It loads device handlers, opens files for read or write operations, deletes and renames files, and creates new files. The Command String Interpreter (the use of which is described in Section 2.5) is part of the USR and can be accessed by any program to interpret device and file I/O information.

This page intentionally blank.

### 2.3.4 Device Handlers

Device handlers for the RT-11 system perform the actual transfer of data to and from peripheral devices. New handlers can be added to the system as files on the system device and can be interfaced to the system by modifying a few monitor tables (see the RT-11 *Software* Support Manual, DEC-11-ORPGA-B-D for instructions on how to interface a new handler to the RT-11 monitor).

## 2.4 GENERAL MEMORY LAYOUT

When the RT-11 System is first bootstrapped from the system device, memory is arranged as shown in the left diagram of Figure 2-1 (this is the case for either the Single-Job or Foreground/Background Monitor, since no foreground job exists yet). The background job is the RT-11 module KMON.

When an RT-11 foreground job is initiated (via the monitor FRUN command, Section 2.7.5.1), room is created for the foreground job to be loaded by decreasing the amount of space available to the background job. The memory maps in Figure 2-1 illustrate the system layout before and after a foreground job is loaded. (Refer also to Chapter 6, Section 6.5.)

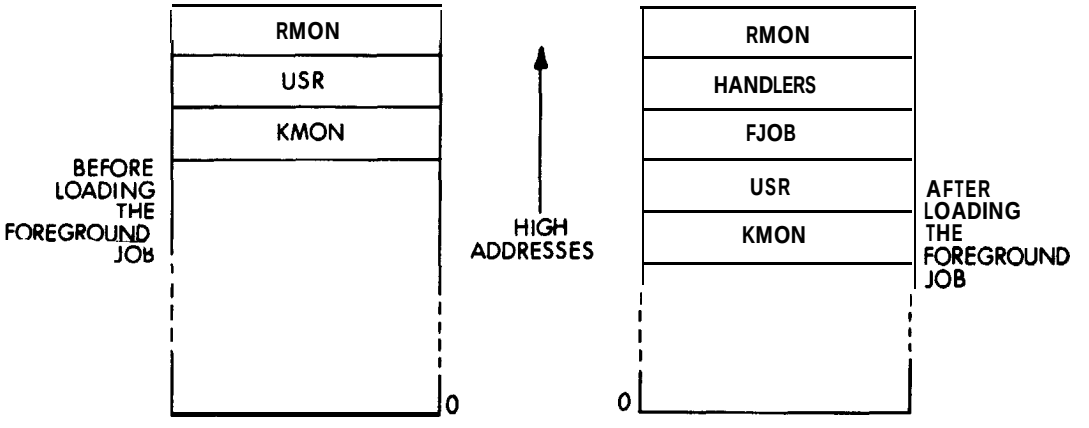



Figure 2-1
RT-11 System Memory Maps

As shown in the figures, the process of loading a foreground job requires that the USR and KMON be physically moved. Once a foreground job is running, it is possible to communicate with either the background or foreground job via special commands (described in Section 2.7). All of the terminal support functions described in Section 2.6 are available under both the Single-job and F/B Monitors.

In addition to FRUN, other monitor commands can alter the memory map; these are LOAD, UNLOAD, GT ON, and GT *OFF*. LOAD causes device handlers to be made resident until an UNLOAD command is performed. UNLOAD deletes handlers which have been loaded. GT ON and GT OFF cause terminal Service to utilize the VT-11 display hardware. Figure 2-2 illustrates the placement of display modules and device handlers in memory following the GT *ON*, LOAD, and FRUN commandst

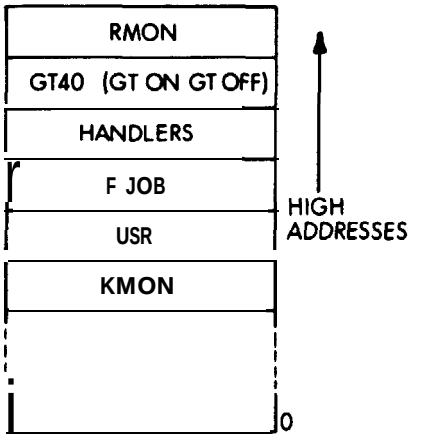



Figure 2-2
RT-11 Memory Map **(GT40)**

RT-11 **maintains** a free memory list to manage memory. Thus, when a handler **is** unloaded, the space the handler occupied **is** returned to the free memory list and **is reclaimed** by the background.

### 2.4.1 Component **Sizes**

Following are the approximate **sizes** (in words) of the components for RT-11, Version 2C **(sizes** reflect **RK).**

| | **F/B** | Single-job |
|---|---|---|
| RMON | 3575 **(10)** | 1703 **(10)** |
| **USR** | 2050 **(10)** | 2050 (10) |
| KMON | **1800(10)** | 1540 (10) |

In the **F/B** System, **the** background area must always be **large** enough to hold KMON and USR **(3.9K** words). The following list indicates the total space available for the loaded **device** handlers, the foreground job, and the display handler. Note that the low memory **area** from **0-477 is** never used for executable programs. (These **sizes** also allow room for the **3.5K RMON).**

| **Machine size** (words) | **Space** available (words) |
|---|---|
| 16K | **8.5K** |
| 24K | **16.5K** |
| **28K** | **20.5K** |

With the Single-Job Monitor, **RMON requires** only **1.67K. The** follcwing list **shows** the **amount** of space available to **users** with the Single-Job **Monitor:**

| Machine size (words) | Program space available (words) |
|---|---|
| 8K | 6K |
| 16K | 14K |
| 24K | 22K |
| 28K | 26K |

## 2.5 ENTERING COMMAND INFORMATION

Once either monitor has been loaded and a system program started, the user must enter the appropriate command information before any operation can be performed.

In most cases, the Command String Interpreter immediately prints an asterisk at the left margin. The user must then type a command string in the general format:

```
OUTPUT=INPUT/SWITCH
```

(A few system programs -- EDIT, PATCH, PATCHO -- require that this command information be entered in a slightly different format. Complete instructions are provided in the appropriate chapter.)

In all cases, the format for OUTPUT is:

**dev:filnam.ext[n],...dev:filnam.ext[n]**

INPUT is:

**dev:filnam.ext,...dev:filnam.ext**

and **SWITCH** is:

**/s:oval** or **/s!dval**

wheret

| | |
|---|---|
| dev: | in each case is an optional two to three-Character name from Table 2-2 whose usage conforms to the NOTE below. |
| filnam.ext | in each case is the name of a file (consisting of one to six alphanumeric characters followed optionally by a dot and a zero to three-Character extension). As many as three output and six input files may be allowed. |
| [n] | is an optional declaration of the numberof blocks (n) deeired for an output file. n is a decimal number (<65,535) enclosed in square brackets immediately following the output filnam.ext to which it applies. |
| /s:oval or /s!dval | is one or more optional switches whose functions vary according to the program in use (refer to the switch option table in the appropriate chapter). oval is either an octal numberor one to three alphanumeric characters (the first of which must be alphabetic) which will be converted to radix-50 (see Section 5.5.4 ofthe MACRO chapter). dval is a decimal value preceded by an exclamation point. |

Throughout this manual, the /s:oval construction is used; however, the /s:dval format is always valid. Generally, these switches and their associated values, if any, should follow the device and filename to which they apply.

If the same switch is to be repeated several times with different values (e.g., /L:MEB/L:TTM/L:CND to MACRO) the line may be abbreviated as /L:MEB:TTM:CND; octal, RAD50, and decimal values may be mixed.

= if required, is a delimiter that separates the output and input fields. The < sign may be used in place of the = sign. The separator can be omitted entirely if there are no output files.

**NOTE**

As illustrated in the general format of a command line, the command line consists of an output list, a separator (= or <), and an input list. Omission of a device specification in either the input or output list is handled as follows:

DK: is assumed if the first file in a list has no explicit device. DK (or the device associated with the first file) is default until another device is indicated; that device then becomes default until a new one is used, and so on. If the following command is entered, for example, to MACRO:

```
*DT1:FIRST.OBJ,LF:=TASK.1,RK1:THSK.2,TASK.3
```

it is interpreted as though all devices had been indicated as followsr

```
*DT1 :FIRST.OBJ, LF:=DK:TASK.1,RK1:TASK.2,RK1:TASK.3
```

## 2.6 KEYBOARD COMMUNICATION (KMON)

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

The special functions of certain terminal keys used for communication with the Keyboard Monitor are explained in Table 2-4. Note that in the F/B system, the Keyboard Monitor always runs as a background job.

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

**Table** 2-4
**Special** Function Xeys

| Key | Function |
|---|---|
| CTRL A | Valid when the monitor GT ON command has been typed and the display ie in use. The **command** does not **echo** on the terminal. It ie used after a CTEL S has been typed to effectively **page** output. Console output is permitted to resume until the **screen is** completely f illed; text previously displayed **is scrolled** upward off the Screen. CTEL A has no special meaning **if** GT ON **is** not in effect **or** if a SET TTY NOPAGE command has been **given (see** Section 2.7.2.8). |
| CTFU **B** | Undar the **F/B** Monitor **echoes** B> on the terminal **(unless** output **is** already coming from the background job) and **causes** all keyboard input to be directed to the background job. At least one line of output will **be** taken from the background job (the foreground job has **priority,** and control will revert to it **if** it has output). All typed input will be directed to the background job until control **is** redirected to the foreground job (via **CTRL** F). CTEL B has no special meaning when used **under** a Single-Job Monitor **or** when a SET TTY NOFB **command** has been fssued (see Section 2.7.2.8). |
| **CTRL** C | CTRL **C** echoes as ^C on the terminal and is used to interrupt program execution and return control to the keyboard monitor. If the **program** to be interrupted is waiting for terminal input, **or** is using the TT handler for input, typing one CTRL C is **sufficient** to interrupt **execution;** in all other **cases,** two CTRL Cs are **neces-**sary. Note that **under** the **F/B** Monitor, the job **which** is currently receiving input will be the jobthatisstopped (determined by whether a CTRL F or CTRL B was **most re-**cently typed). To ensure that the command is directed to the proper job, type CTRL B or CTRL F before **typing** CTRL C. |
| **CTRL** E | Valid when the monitor GT ON command has been typed and the display **is** in use. The command does not **echo** *on* the terminal, but **causes** all terminal output to appear on both the diaplay **screen** and **the console** terminal simultaneously. A **second CTRL** E **disables console** terminal output. **CTRL** E has no special meaning **if** GT **ON is** not in effect. |
| **CTRL** F | **Under** the **F/B** Monitor **echoes** F> on the terminal and instructs that all keyboard input be directed to the foreground job and all output be taken **from** the foreground job. If no foreground job exfsts, F? **is** printed and control **is** directed to **the** background job. **Otherwise,** control remains with the foreground job until redirected to the background job (via **CTRL B) or** until the foreground job terminatea. CTRL F has no special meaning when used **under** a Single-Job Monitor, **or** when a SET **TTY** NOFB command has been used **(see** Section 2.7.2.8). |

Table 2-4 (Cont.)
Special Function Keys

| Key | Function |
|---|---|
| CTRL O | Echoes ↑O on the terminal and causes suppresaion of teleprinter output while continuing program execution. Teleprinter output is re-enabled when one of the following occurs:<br>1. A second CTRL 0 is typed,<br>2. A retum to the monitor occurs, or<br>3. The running program issues a .RCTRLO directive (see Chapter 9). (RT-11 system programs reset CTRL 0 to the echoing state each time a new command string is entered.) |
| CTRL Q | Does not echo. Resumes printing characters on the terminal from the point at which printing was previously stopped (via CTRL S). CTRL Q has no special meaning if a SET TTY NOPAGE connnand has been used (see Section 2.7.2.8). |
| CTRL S | Does not echo. Temporarily suspends output to the terminal until a CTRL Q is typed. If GT ON is in effect, each subsequent CTRL A causes output to proceed until the screen has been refilled once. This feature allows users with high-speed terminal8 to fill the display screen, stop output with CTRL S, read the Screen, and then continue with CTRL Q or CTRL A. (Typing CTRL C in this case also continues output.) Under the F/B Monitor, CTRL S has no special meaning if a SET TTY NOPAGE has been used. |
| CTRL U | Deletes the current input line and echoes as ↑U followed by a carriage retum at the terminal. (The current line is defined to be all characters back to, but not including, the most recent line feed, CTRL C or CTRL Z.) |
| CTRL Z | Echoes ↑Z on the terminal and terminates input when used with the terminal device handler (TT). The CTRL Z itself does not appear in the input buffer. If TT is not being used, CTRL Z has no special meaning. |
| RUBOUT | Deletes the last character frcm the current line and echoes a backslash plus the character deleted. Each succeeding RUBOUT deletes and echoes another Character. An enclosing backslash is printed when a key other than RUBOUT is typed. This erasure is done right to left up to the beginning of the current line. |

### 2.6.1 Foreground/Background Terminal I/O

It is important to note that console input and output under F/B are independent functions; input can be typed to one job while output ie printed by another. The user may be in the process of typing input to one job when the other job ie ready to print on the terminal. In this case, the job which is ready to print interrupts the user and prints the message on the terminal; input control is not re-directed to this job, however, unlese a CTRL B or CTRL F is explicitly typed. ff Input is typed to one job while the other has output

control, echo of the input is suppressed until the job accepting input gains output control; at this point all accumulated input is echoed.

If the foreground job and background job are both ready to print output at the same time, the foreground job has priority. Output from the foreground job prints until a line feed ie encountered, at which 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.

### 2.6.2 Type-Ahead

The monitor has a type-ahead feature which allows terminal input to be entered while a program is executing. For examplet

```
.R FIF
*DT1:TAPE=PR:
DT1:/L
*13-FEB-74
TAPE          78 13-FEB-74
 486 FKEE BLOCKS
```

While the first command line is executing, the second line (DT1:/L) is entered by the user. This terminal input is stored in a buffer and used when the first Operation has completed.

If a single CTRL C ie typed while in this mode, it is put into the buffer. The program currently executing exits when a terminal input request needs to be satisfied. A double CTRL C returns control to the monitor immediately.

If type-ahead input exceeds 80 characters, the terminal bell rings and no characters are accepted until part of the type-ahead buffer is used by a program or characters are deleted. No input is lost. Type-ahead is particularly useful in specifying multiple command lines to system programs, as shown in the preceding example. ff a job ie terminated by typing two CTRL C's, any unprocessed type-ahead is discarded.

**NOTE**

ff type-ahead is used in conjunction with EDIT or BASIC, there is no terminal echo of the characters but they are stored in the buffer until a new command is needed. The characters are echoed only when actually used by the program.

## 2.7 KEYBOARD COMMANDS

Keyboard commands allow the user to communicate with the monitor. Keyboard commands can be abbreviated; optional characters in a command are delimited (in this section only) by braces. Keyboard commands require at least one space between the command and the first argument. All command linea are terminated by a carriage return.

All canmands, with the exception of those described in.Section 2.7.5, may be used under either the Single-Job or F/B Monitor. The commands described in Section 2.7.5 apply only to the F/B Monitor.

**NOTE**

Any reference made to "the background job" applies as well to the Single-Job Monitor, since the background job in a F/B system is equivalent to the single-job environment in its normal state.

### 2.7.1 Commands to Control Terminal I/O (GT ON and GT OFF)

GT ON/GT OFF

The GT ON and GT OFF commands are used to enable and disable the scroller (VT-11 display hardware). GT ON causes the display screen to replace the console as the terminal output device. Switch options allow the user to control the number of lines to appear on the screen and to position the first line vertically. output appears an the display in the same format as it would on the console (i.e., output, text, and commands are displayed in the order in which they occur). GT ON is not permitted in an 8K configuration.

The form of the GT ON command is:

GT ON{/L:n}{/T:n}

wherc:

**/L:n** represents an optional switch setting indicating the number of lines of text to diaplay; the suggested range is:

| | |
|---|---|
| 12" screen (GT40, DEClab) | 1<=n<=37 octal (31 decimal) |
| 17" Screen (GT44) | 1<=n<=50 octal (40 decimal) |

**/T:n** represents an optional switch setting indicating the top position of the scroll diaplay; the suggested range is:

| | |
|---|---|
| 12" screen (GT40, DEClab) | 1<=n<=1350 octal (744 decimal) |

| | |
|---|---|
| **17" screen (GT44)** | **1<=n<=1750** octal (1000 decimal) |

ff no switches are specified, a test for the **screen size is** performed and default values are automatically assigned as **follows:**

| | |
|---|---|
| **12" screen (GT40, DEClab)** | /Lt37 (31 decimal)<br>**/T:1350** (744 decimal) |
| **17" screen (GT44)** | **/L:50** (40 decimal)<br>**/T:1750** (1000 decimal) |

Line length is always set to 72 for 12" **screen** and 80 for 17" **screen.** Once the display has been activated with the GT ON command, CTRL A, CTRL S, CTRL E and CTRL Q **can** be used to control Scrolling behavior. These **commands** are described in **Section** 2.6.

NOTE

ODT is one exception to the use of GT ON. This **system program** has **its** own terminal handler and **cannot** make use of the **display;** output will appear only on the **console** terminal whenever ODT **is** running .

The GT OFF **command** clears the display and resumes output on **the** teleprinter. The command format **is:**

GT OFF

If GT ON and GT OFF are used when no display hardware exists **or** when a foreground job **is active,** the ?ILL **CMD?** measage **is** printed.

## 2.7.2 Commands to **Allocate** System Resources

**DATE**

2.7.2.1 **DATE** Command - The DATE command enters the indicated date to the **system.** This date is then assigned to newly created files, *new* **device** directory **entries (which** may be listed with PIP), and **listing** output **until** a new DATE command is issued.

The form of the command ist

**DAT{E}** **{dd-mmm-yy}**

where **dd-mmm-yy is** the day, month and year to be entered. dd is a decimal **number** in **the range 1-31; mmm is** the **first** three **characters** *of* the name of the month, and yy **is** a decimal **number** in the range 73-99. If no argument is given, the **current** date **is** printed.

Examplest

| | |
|---|---|
| . DATE ZI-FEE-74 | Enter the date 21-FEB-74 as the current system date. |
| , DAT<br>21-FEB-74 | Print the current date. |

ff the date is entered in an incorrect format, the ?DAT? error message is printed.


2.7.2.2 TIME Command - The TIME command allows the user to find out the current time of day kept by RT-11 or to enter a new time of day. If no KW11-L clock is present on the System, the ?NO CLOCK? error message is generated. If the time is entered in an incorrect format, the ?TIM? message is printed.

The fonn of the command is:

TIM {E} {hh:mm:ss}

where hh:mm:ss represents the hour, minute, and second. Time is represented as hours, minutes, and seconds past midnight in 24-hour format (e.g., 1:25:00 P.M. is entered as 13:25:00). ff any of the arguments are omitted, 0 is assumed. If no argument is given, the current time of day is output.

Examplest

| | |
|---|---|
| . TIM8:15:23 | Sets the time of day to 8 hours, 15 minutes and 23 seconds. |
| .TIM<br>08:25:27 | Approximately 10 minutes later, the TIME command Outputs this time. |
| . TIME 18:5 | Sets the time of day to 18:05:00. |

Under the F/B Monitor, after the time reaches 24:00, the time and date will be reset when the user next issues a TIME command (or .GTIM programmed request). Time and date are not reset under the Single-Job Monitor. Month and year are not updated under either monitor.

The clock rate is initially set to 60-cycle. Consult the RT-11 System Generation Manual if conversion to a 50-cycle rate is necessary.

## INITIALIZE

2.7.2.3 INITIALIZE Command - The INITIALIZE command ie used to reset several background system tables and do a general "clean-up" of the background area; it has no effect on the foreground job. In particular, this command makes non-resident those handlers which were not loaded (via LOAD), purges the background's I/O channels, disables CTRL 0, performs a hard reset , clears locations 40-53, resets the KMON stack pointer, and under the F/B monitor performs an .UNLOCK.

Under the Single-Job Monitor a RESET instruction is done (see Chapter 9). Under the F/B Monitor, I/O is stopped by entering each busy device handler at a special abort entry point.

The form of the command ist

IN{ITIALIZE}

The INITIALIZE command can be used prior to running a user program, or when the accumulated results of previously issued GET commands (see Section 2.7.3.1) are to be discarded.

**Example:**

```
.IN              Initializes system background job
.R PROG
```

## ASSIGN

2.7.2.4 ASSIGN Command - The ASSIGN Command assigns a user-defined (logical) name as an alternate name for a physical device. This is especially useful when a program refers to a device which is not available on a certain System. Using the ASSIGN command, I/O can be redirected to a device which is available. Only one logical name can be assigned per ASSIGN command, but several ASSIGN commands (14 maxiraunt) can be used to assign different names to the same device. This command is also used to assign *FORTRAN* logical units to device names.

The form of the command isr

**ASS {IGN} {{dev}:udev}**

wheret

dev — is any Standard RT-11 (physical) device name (refer to Table 2-2) with the exception of DK and SY.

udev — is a 1-3 Character alphanumeric (logical) name to be used in a program to represent dev (if more than three characters are given, only the first three are actually used). DK and SY may be used as loqical device names.

: — is a delimiter character (can be a colon, equal sign, and, if eeparating physical and logical devices, space).

The placement of the delimiter is very important in the ASSIGN command; it must be placed exactly as shown in the following examples:

| | |
|---|---|
| ASSIGNDT1 INP | Physical device DT1 ie assigned the logical device name INP. Whenever a reference to INP: is encountered, device DT1: is used. |
| . HSSIGN DT3 :DK | Physical device name DT3 is assigned the default device name DK. Whenever DK ie rererenced or defaulted to, DT3 is used. (Note that the initial assignment of DK is thus changed.) |
| . ASSIGN LP=9 | FORTRAR logical unit 9 becomes the physical device name LP. All referencea to unit 9 use the line printer for output. |

Assignment of logical names to logical names is not allowed.

ff only a logical device name is indicated in the command line, that particular assignment (only) is removed. Thus:

| | |
|---|---|
| . ASSIGN :9 | Deassigns the logical name 9 from its physical device (LP, in the case above). |
| . ASSIGN =DK | Removes assignment of logical name DK from its physical device (DT3, in the case above). |

If neither a physical device name nor a logical device name is indicated, all assignments to all devices are removed.

| | |
|---|---|
| . ASSIGN | All previous logical device assignments are removed. |

## CLOSE

2.7.2.5 *CLOSE* Command - The CLOSE command causes all currently open output filea in the background job to become permanent files. ff a tentative open file is not made permanent, it will eventually be deleted. The CLOSE command is most often used after CTRL C has been typed to abort a background job and to preserve any new files that job had open prior to the CTRL C; it has no effect on a foreground job.

The form of the command ist

CLO{SE}

The CLOSE command makes temporary directory entries permanent.

Exampler

```
 R EDIT
*EWTEXT$$
*IABCD$$
*^C

 CLOSE
```

The Editor has a temporary file open (TEXT), which is preserved by .CLOSE.

## LOAD

2.7.2.6 **LOAD** Command - The **LOAD** command is used to make a device handler resident for use with background and foreground jobs. Execution is faster when a handler is resident, although memory area for the handler must be allocated. Any device handler to be used by a foreground job must be loaded before it can be used.

The form of the command is:

LOA{D} dev{,dev=B}{,dev=F,...}

wheret

| | |
|---|---|
| dev | represents any legal RT-11 device name. |
| = | represents a delimiter, denoting device ownership. |
| B | represents the background job. |
| F | represents the foreground job. |

The dev=F (and dev=B) construction is valid only under the Foreground/BackgroundSystem. When used under the Single-Job Monitor, the ?ILL EV? error message occurs.

A device may be owned exclusively by either the foreground or background job. This may be used, for example, to prevent ths I/O of two different jobs from being intermixed on the same non-file structured device. For example:

```
·LOAD PP=B,PR,LP=F
```

The Papertape punch belongs to the background job while the paper tape reader 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.

Different units of the same random-access device Controller may be owned by different jobs. Thus, for example, DT1 may belong to the background while DT5 may belong to the foregrouna job. If no ownership is indicated, the device is available for public use.

To change ownership of a device, another LOAD command may be used; it is not necessary to first UNLOAD the device. For example, if RK1 has been assigned to the foreground job as in the example above, the commandr

```
.LOA RK1=B
```

reassigns it to the background job.

The system unit of the system device cannot be assigned ownership, and attempts to do so will be ignored. Other units of the same type as the system device, however, can be assigned ownership.

LOAD is valid *for* use with user-assigned names. For example:

```
.ASSIGN RK2:XY
.LOA XY=F
```

If the Single-Job, DECtape-based Monitor is being used, loading the necessary device handlers into memory can significantly improve the throughput of the System, since no handlers need to be loaded dynamically (in other words, they need not be loaded, as required, from the DECtape).

## UNLOAD

2.7.2.7 UNLOAD Command - The UNLOAD command is used to make handlers that were previously LOADed non-resident, *freeing the* memory they were using.

The form of the command ist

UNL{OAD} dev {,dev,...}

wheret

dev represents any legal RT-11 device name.

UNLOAD clears ownership for all units of an indicated device type. For example, typing:

```
UNL RK2
```

clears all units of RX. (A request to unload the system device handler clears ownership for any assigned units for that device, but the handler remains resident.)

Any memory freed is returned to a free memory list and eventually reclaimed for the background job after the UNLOAD command is given. UNLoAD is not permitted if the foreground job is running. such an action might cause a handler which is needed by the foreground job to become non-resident.

Example:

```
UNLORD LP, PP
```

The lineprinter and paper tape punch handlers are releaaed and the area which they used is freed.

A special function of this command is to remove a terminated foreground job and reclaim memory, since the space occupied by the foreground job is not automatically returned to the free memory list when it finishes. In this instance, the device name FG is used to specify the foreground job. For examplet

```
UNL FG
```

FG can be mixed with other device names.

Iiowever, if, for example, DT2 has been assigned the name FG and loaded into memory as followsr

```
.ASSIGN DT2:FG
.LOAD FG
```

the commandt

```
.UNLOAD FG
```

causes the foreground job, not DT2, to be unloaded. To unload DT2, this command must be typed:

```
UNLOAD DT2
```

SET

2.7.2.8 SET Command - The SET command is used to change device handler characteristics and certain system configuration Parameters.

The form of the command is:

SET devt {NO}option=value {,{NO}option=value,...}

where :

| | |
|---|---|
| devr | represents any legal RT-11 physical device name (and in addition, TTY and USR). |
| {NO}Option | is the feature or characteristic to be altered. |
| =value | is a decimal nurober required in some cases. |

A space may be used in place of or in addition to the colon, equal sign, or comnaa. Note that the device indicated (with the exception of TTY and USR) must be a physical device name and is not affected by logical device name assignments which may be active. The name of the characteristic or feature to be altered must be legal for the indicated device (see Table 2-5) and may not be abbreviated.

The SET command locates the file SY:dev.SYS and permanently modifies it. No modification is done if the command entered is not completely valid. If a handler has already been loaded when a SET command is issued for it, the modifications will not take effect until the handler is unloaded and a fresh copy called in from the system device.

Table 2-5 lists the system characteristics and Parameters which may be altered (those modes designated as "normal" are the modes as set in the distribution copies of the drivers):

Table 2-5
SET Command Options

| Device | Option | Alteration |
|---|---|---|
| LP | CR | Allows carriage returns to be Sent to the printer. The CR option should be set for any FORTRAN program using formatted I/O, to allow the overstriking capability for any line printer, and when using the LS11 or LP05 line printers (since the last line in the buffer mav otherwise be lost). This is the normal mode. |
| LP | NOCR | Inhibits sending carriage returns to the line printer. The line printer Controller causes a line feed to perform the functions of a carriage return, so using this option produces a significant increase in printing speed on LP11 printers. |
| LP | CTRL | Causes all characters, including nonprinting control characters, to be passed to the line printer. This mode may be used for LS11 line Printers. (Other line printers will print space for control characters.) |

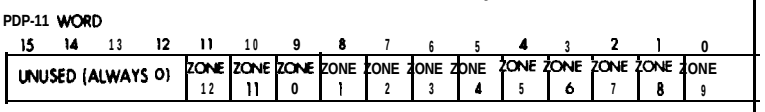
(continued on next page)

Table 2-5 (Cont.)
SET Command Options

| Device | Option | Alteration |
|---|---|---|
| LP | NOCTRL | Ignores nonpfinting control characters. This is the normal mode. |
| LP | FORM0 | Causes a form feed to be issued before a request to print block zero. This is the normal mode. |
| LP | NOFORM0 | Turns off FORM0 mode. |
| LP | HANG | Causes the handler to wait for user correction if the line printer is not ready or becomes not ready during printing. This is the normal mode.<br><br>New users should note that when expecting output from the line printer and it appears as though the system is not responding or is in an idle state, the line printer should be checked to see if it is on and ready to print. |
| LP | NOHANG | Generates an immediate error if the line printer is not ready. |
| LP | LC | Allows lower case characters to be sent to the printer. This option should be used if the printer has a lower case character set. |
| LP | NOLC | Causes lower case characters to be translated to upper case before printing. This is the normal mode. |
| LP | WIDTH=n | Sets the line printer width to n, where n is a number between 30 and 255. Any characters printed past column n are ignored. The NO modifier is not permitted. |
| CR | CODE=n | Modifies the card reader handler to use either the DEC 026 or the DEC 029 card codes (refer to Appendix H). n must be either 26 or 29. The NO modifier is not permitted. |
| CR | CRLF | Causes a carriage return/line feed to be appended to each card image. This is the normal mode. |
| CR | NOCRLF | Transfers each card image without appending a carriage return/line feed. |
| CR | HANG | Causes the handler to wait for user correction if the reader is not ready at the start of a transfer. This is the normal mode. |
| CR | NOHANG | Generates an immediate error if the device ie not ready at the start of a transfer. Note that the handler will wait regardless of how the option ie 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 yet been read). |

 (continued on next page)

Table 2-5 (Cont.)
SET Command Options

| Device | Option | Alteration |
|---|---|---|
| CR | IMAGE | Causes each card column to be stored as a 12-bit binar-y number, one column per word. The CODE option has no effect in IMAGE mode. The format of the 12-bit binary number is:<br>  <br><br>This format allows binary card images to be read and is especially useful if a special encoding of punch combinations is to be used. Mark-sense cards may be read in IMAGE mode. |
| CR | NOIMAGE | Allows the normal translation (as specified by the CODE Option) to take place; data is packed one column per byte. Invalid punch combinations are translated into the error Character, ASCII "\" (backslash), which is octal code 134. This is the normal mode. |
| CR | TRIM | Causes trailing blanks to be removed from each card read. It is not recommended that TRIM and NOCRLP be used together since card boundaries will be difficult to find. This is the normal mode. |
| CR | NOTRIM | Transfers a full 80 characters per card. |
| CT | RAW | Causes the cassette handler to perform a read-after-write check for every record written, and retry if an output error occurred. If three retries fail, an output error is detected. |
| CT | NORAW | Causes the cassette handler to write every record directly without reading it back for verification. This significantly increases transfer rates at the risk of increased error rates. Normal mode is NORAW. |
| The folloa ng Options, with the exception of HOLD/NOHOLD and COPY/NOCOPY, are available in the Foreground/Background System only; HOLD/NOHOLD and COPY/NOCOPY are available in both Systems. These Options are not permanent, and must be reissued whenever the monitor is re-bootstrapped. They can be made permanent by modifying the monitor as described in Chapter 2 of the RT-11 Software Support Manual. (Note that the device specification is TTY, not TT, because the handler itself is not changed.) | | |
| TTY | COPY | Enables use of the auto-print mode of the VT50 copier option, if present. The command is a no-op for any terminal other than the VT50, but a "}" Character may be printed on the terminal. Consult the VT50 Video Terminal Programmer's Manual for more information. |
| TTY | NOCOPY | Disables use of the auto-print mode of the VT50 copier Option, if present. The command is a no-op for any terminal other than the VT50, but a "^" character may be printed on the terminal. This ie the normal mode. |



(continued on next page)

**Table 2-5 (Cont.)**
SET command Options

| Device | Option | Alteration |
|---|---|---|
| TTY | CRLF | Causes the monitor to issue a carriage return/line feed on the console terminal whenever it attempts to type past the right margin (as set by the WIDTH option). This is the normal mode. |
| TTY | NOCRLF | Causes no special action to be taken at the right margin. |
| TTY | FB | Causes the monitor to treat CTRL B and CTRL F characters as background and foreground program control characters and does not transmit them to the user program. This is the normal mode. |
| TTY | NOFB | Causes CTRL B and CTBL F to have no special meaning.<br>NOTE<br>SET TTY NOFB is issued to KMON, (which runs as a background job) and disables all communication with the foreground job. To enable communication with the foreground job, issue the command SET TTY FB. |
| TTY | FORM | Indicates that the console terminal is capable of executing hardware form feeds. |
| TTY | NOFORM | Causes the monitor to simulate form feeds by typing eight line feeds. This is the normal mode. |
| TTY | HOLD | Enables use of the hold screen mode of operation for the VT50 terminal. The command is a no-op for any terminal other than the VTSO, but a "[" character may be printed on on the terminal. The command is valid for F/B and Single-Job Monitors. Consult the VT50 Video Terminal Programmer's Manual for more information. |
| TTY | NOHOLD | Disables use of the hold screen mode of operation for the VT50 terminal. The command ie a no-op for any terminal *other* than the VT50, but a "\" Character may be printed on the terminal. This is the normal mode. |
| TTY | PAGE | Causes the monitor to treat CTBL S and CTRL Q characters as terminal output hold and unhold flags, and does not transmit them to the user program. This is the normal mode. |

(continued on next Page)

Table 2-5 (Cont.)
SET Command Options

| Device | Option | Alteration |
|---|---|---|
| TTY | NOPAGE | Causes CTRL S and CTRL Q to have no special meaning . |
| TTY | SCOPE | Causes the monitor to echo RUBOUTs as backspace-space-backspace. This mode should be used when the console is a VT05/VT50 or when GT ON is in effect. |
| TTY | NOSCOPE | Causes the monitor to echo RUBOUTs as backslash followed by the character deleted. This is the normal mode. |
| TTY | TAB | Indicates that the console terminal ie capable of.executing hardware tabs. |
| TTY | NOTAB | Causes the monitor to simulate tab stops every eight positions. The normal mode is NOTAB. VT05/VT50 terminale generally have hardware tabs. |
| TTY | WIDTH:n | Sets the width of the console terminal to n positions, for the use of the CRLF option. n must be in the range 30-255 (decimal). The width is initially set to 72. |

The following variant of the SET command is used to prevent the background job from ever placing the USR in a swapping state (note that USR replaces a device specification in the command line):

SET USR {NO} SWAP

This is useful when running on a DECtape based System, or when running a foreground job which requires the USR but has no memory allocated into which to read it. When the monitor is bootstrapped, it ie in the SWAP condition, i.e., the background may place the USR in a swapping state via a SETTOP.

The Single-Job Monitor behaves as though the following options are sett NOTAB, NOFORM, PAGE, NOCRLF, NOSCOPE, NOHOLD.

### 2.7.3 Commands to Manipulate Memory Images

**GET**

2.7.3.1 GET Command - The **GET** command (valid for use **with** a **background** job **only)** loads the specified memory image file (not ASCII or object) into memory from the indicated device.

The fonn of the GET **command is:**

GE{T} devrfilnam.ext

where:

| | |
|---|---|
| devt | represents any legal RT-11 device name. If a device **is** not specified, DK: ie assumed. Note that devices MT and CT are not block-replaceable devices and therefore cannot **be** used in a GET command. |
| **filnam.ext** | represents a valid RT-11 filename and extension. **If** an extension ie not specified, the extension .SAV **is assumed.** |

The GET command **is** typically used to load a **program** into memory for modificatfon **and/or** debugging. The GET **command can** also be used in conjunction with the Base, Examine, Deposit, and START commands to test patches, and **can** be used with **SAVE** to make patches permanent. Multiple **GETs can** be used to **combine** programs. Thust

| | |
|---|---|
| . GET ODT. SAV | **Loads** ODT into memory |
| GET PROG | **Loads** PROG.SAV into memory with ODT |
| START (ODTs starting address) | Starts **execution** with ODT **(see** Chapter 8). |

The GET command cannot be used to load overlay Segments **of** programs; it may only be used to load the root Segment **(that** part **which** will not be overlaid; refer to Chapter 6, Linker).

Multiple **GETs can** be used **to** build a memory image of several programs. ff identical locations are required by any of the programs, **the** latar programs overlay the previous ones.

**Examples:**

| | |
|---|---|
| GET DT3:FILE1. SAV | **Loads** the file **FILE1.SAV** into memory **from DECtape** unit 3. |
| GET NAME1 | Loads the file **NAME1.SAV** from device DK. |

**BASE**

2.7.3.2 Base **Command -** The B **command** sets a relocation base. This relocation base is added to the address specified in subsequent **Examine or** Deposit **commands** to **obtain** the addrees of the location to be referenced. This c**ommand** is useful when referencing **linked modules** with the Examine and Deposit **commands.** The base address **can** be **set** to the address where the module of interest is loaded. The **form** of the command ist

**B** {**location**}

wherer

location represents an octal address used as a base address for subsequent Examine and Deposit **commands.**

**NOTE**

A space must follow the B command even if an address **is** not specified (the **B<space> command** is equivalent to B 0).

Any non-octal digit terminates an address. If location ie odd, it is rounded down by one to an even address.

The base is cleared whenever **user program execution** ie initiated.

**Examples:**

| | |
|---|---|
| .BΔ | Sets **base** to 0 (Δ represents space). |
| .B 200 | Sets **base** to 200. |
| B 201 | Sets **base to 200.** |

## EXAMINE

2.7.3.3 Examine Command - The E command prints the contents of the specified location(s) in octal on the console terminal. The form of the Examine command ist

E location m{-location n}

wherer

location represents an octal address which is added to the relocation base value (the value set by the B Command) to get the actual address examined. Any non-octal digit terminates an address. An odd address is truncated to become an even address.

If more than one location ie specified (location m-location n), the contents of location m through location n inclusive are printed. The second location specified (location n) must not be less than the first location specified, otherwise an error message is printed. If *no* location is apecified, the contents of location 0 are printed. Examination of locations outside the background area ie illegal.

Examples:

```
.E 1000
127401
```

Prints contents of location 1000 (added to the base value if other than 0).

```
.E 1001-1012
127401 007624 127400 000000 000000 000000
```

Prints the contents of locations 1000 (plus the base value if other than 0) through 1013.

## DEPOSIT

2.7.3.4 Deposit Command - The Deposit command deposits the specified value(s) starting at the location given.

The form of the command is:

D location-value1{value2,...valuen}

wheret

location represents an octal address which is added to the relocation base value to get the actual addrees where the values are deposited. Any non-octal digit is accepted as a terminator of an address.

value represents the new contents of the location. 0 is assumed if a value is not indicated.

ff multiple values are specified (valuel,...,valuen), they are deposited beginning at the location specified. The DEPOSIT canmand accepts word or byte addresses but executes the command as though a word address was specified. An odd address ie truncated by one to an even address. All values are stored as word quantities.

Any character that is not an octal digit may be used to separate the locations and values in a DEPOSIT command. However, two (or more) non-octal separators cause 0's to be deposited at the location specified (and those following). For example:

.D 56,,, Deposits 0's in locations 56, 60, and 62.

The user should be aware of situations like the above, which causes system failure since the terminal vector (location 60) is reroed.

An error results when the address specified references a location outside the background job's area.

Examples:

| | |
|---|---|
| D 1000=3705 | Deposits 3705 into location 1000 |
| . B 1000 | Sets relocation base to 1000 |
| .D 1500=2503 | Pute 2503 into location 2500 |
| . B 0 | Resets base to 0 |

## SAVE

2.7.3.5 SAVE Command - The SAVE Command writes specified user memory areas to a named file and device in save image format. Memory is written 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).

The SAVE Command does not write the overlay segments of programs; it saves only the root segment (refer to Chapter 6, Linker).

The form of the command is:

SAV{E} dev:filnam.ext (parameters}

wheret

devr represents one of the standard PT-11 block-replaceable device names. If no device is specified, DK ie assumed.

**file.ext** represents the name to be assigned to the file being saved. If the file name is omitted, an error message ie output. If no extension is specified, the extension .SAVis used.

Parameters represent memory locations to be saved. RT-11 transfers memory in 256-word blocks beginning on boundaries that are multiples of 256(decimal). If the locations specified make a block of less than 256 words, enough additional locations are transferred to make a 256-word block.

Parameters can be specified in the following format:

**areal,area2-arean**

**where:**

areal
**area2-arean** represent an octal number (or numbers separated by dashes). It more than one number is specified, the second number must be greater than the first.

The start address and the Job Status Word are given the default value 0 and the stack is set to 1000. ff the user wants to change these or any of the following addresses, he must first use the DEPOSIT connnand to alter them and *then* SAVE the correct areasr

| Area | Location |
|---|---|
| Start address | 40 |
| Stack | **42** |
| JSW | **44** |
| USR address | **46** |
| High address | **50** |
| **Fill characters** | 56 |

If the values of the addresses are changed, it is the user's responsibility to reset them to their default values. See Chapter 9 for more information concerning these addresses.

**Examples:**

```
.SAVE FILE1 10000-11000,14000-14100
```

Saves locations 10000(8) through 11777(8) (11000 starts the first word of a new block, therefore the whole block, up to 12000, ie stored) and 14000(8) through 14777(8) on DK with the name FILE1.SAV.

```
.SAVE DT1:NAM.NEW 10000
```

Saves locations 10000 through 10777 on DT1: with the name NAM.NEW.

```
.D 44:20000
.SAV SY: PRAM 1000-5777
```

Sets the reenter bit in the JSW and saves locations 1000 through 5777.

## 2.7.4 Commands to Start a Program

### RUN

2.7.4.1 RUN Command - The RUN command (valid for use with a background job only) loads the specified memory image file into memory and starts execution at the start address specified in location 40. Under the F/B system, 10 words of user stack area are required to start a user program, and the stack address (location 42) must be initialised to some part of memory where these 10 words will not modify it.

The form of the command ist

RU{N} dev:filnam.ext

where :

**dev:** is any standard device name specifying block-replaceable device. ff dev: is not specified: the device ie assumed to be DK. Note that devices MT and CT are not block-replaceable devices and therefore cannot be used in a RUN command.

**filnam.ext** ie the file to be executed. ff an extension ie not specified, the extension .SAV ie assumed.

The RUN command is equivalent to a GET command followed by a START command (with no address specified).

**NOTE**

If a file containing overlays is to be RUN from a device other than the system device, the handler for that device must be loaded (see Section 2.7.2.6) before the RUN command is issued.

Examples:

| | |
|---|---|
| . RUN DT1:SRCH.SAV | Loads and executes the file SRCH.SAV from DT1. |
| . RUN PROG | Loads PROG.SAV from DK and executes the program. |
| . GET PROG1 | Loads PROG1.SAV from device DK without executing it. Then combines PROG1 and |
| . RUN PROG2 | PROG2.SAV in memory and begins execution at the starting address for PROG2. |

## R

2.7.4.2 R Command - This command (valid for use with the background job only) ie similar to the RUN command except that the file specified must be on the system device (SY:).

The form of the command ist

R filnam.ext

No device may be specified. If an extension is not given, the extension .SAV is assumed.

**Examples:**

. R XYZ. SAV — Loads and executes XYZ.SAV from SY.

. R SRC — Loads and executes SRC.SAV from SY.

## START

2.7.4.3 START Command - The START Command begins execution of the program currently in memory (i.e., loaded via the GET Command) at the specified address. START does not clear or reset memory areas.

The formof the command is:

ST{ART} {address}

wheret

address — is an octal number representing any 16-bit address. If the address is omitted, or if 0 is given, the starting address in location 40 will be used.

ff the address given does not exist or is not an even address, a trap to location 4 occurs. In this case a monitor error message appears. If no address is given, the program's atart address from location 40 is used.

**Examples:**

```
.GETFILE.1
.START 1000
```

Loads FILE.1 into memory and starts execution at location 1000.

```
.GET FILEA
.GETFILEB
,ST
```

Loads FILEA.SAV, then combines FILEA.SAV with FILEB.SAV and starts execution at FILEB's start address.

## REENTER

2.7.4.4 REENTER Command - The REENTER command starte the program at its reentry addrees (the start address minus two). REEWTER does not clear or reset any memory areas and is generally used to avoid reloading the same program for repetitive execution. It can be used to return to a program whose execution was etopped with a CTRL C.

The form of the command ist

RE{ENTER}

If the reenter bit (bit 13) in the Job Status Word (location 44) is not set, the REEWTER Command is illegal.

For most system programs, the REENTER command restart8 the program at the command level.

If desired, the reentry point in a user program can branch to a routine which initializes the tables and stack, fetches device handlers etc., and then continue normal operation.

**Example:**

```
.R PIP
*/F
MONITR.SYS
[directory prints]

:(^C typed)
.
 REENTEH
*
```

CTRL C interrupts the PIP directory listing and transfers control to the monitor level. REENTER returns control to PIP.

2.7.5 Commands Used Only in a Foreground/Background Environment

It is important to note that in order to control execution of a foreground job, the commands in this section must be typed to KMON, which is running as the background job. Thus, for example, to SUSPEND

the foreground job, the user must be sure he is directing input to KMON as follows:

```
F>
    OB typed)
B>
R PIF
*^C
, SUSPEND
```

Foreground job is running. Control ie redirected to the background job and PIP is called (the foreground ie still active). CTRL C stops PIP and starts KMON. The foreground job is suspended. (See Section 2.7.5.2.)


2.7.5.1 FRUN Command - The *FRUN* command ie used to initiate foreground jobs. *FRUN* will only run relocatsble files produced with the Linker /R switch (using the Linker supplied with RT-11, Version 2). Any handlers used by a foreground job must be in memory.

The form of the command ist

FRU{N} dev:file.ext{/N:n}{/S:n}{/P}

where:

dev: represents a block replaceable RT-11 device. If dev: is not specified, DK: ie assumed.

file.ext represents the job to be executed. The default extension for a foreground job is .REL.

/N:n or /N!n represents an optional switch used to allocate n words (not bytes) over and above the actual program size. (If *running a* FORTRAN program, *a* special formula is used to determine n. Refer to Appendix G for this information.)

/S:n or /S!n represents an optional switch used to allocate n words (not bytes) for stack space. Normally, stack space is set by default to 128 words and is placed in memory below the program. To change the stack size, use /S:n; the stack is still placed in memory under theprogram. To relocate the stack area, use an .ASECT (see Chapter 5) to define the start of the *user* stack in location 42. This overrides the /S switch.

/P represents an optional switch (at the end of the FRUN command) for debugging purposes. When the carriage return is typed, FRUN prints the load address of the program, but does not start the

program. The foreground job must be explicitly started with the RSUME command (see Section 2.7.5.3). For example:

```
.FRUN DATA/P
LOADED AT 125444
```

If ODT is used with the foreground job, this feature provides the means for determining where the job actually was loaded.

The programis startedwhen the RSUME command is given, allowing the programmer to examine or mcdify the program before starting it.

ff another foreground job is active when the FRUN command ie given, an error message is printed. ff a terminated foreground job ie occupying memory, that region is first reclaimed. Then if the file indicated is found and will fit *in* memory, the job is installed and started immediately. FRUN destroys the background job's memory image.

**Examples:**

| | |
|---|---|
| . FRUN FI | Runs program F1.REL stored on device DK. |
| . FRU DT1:F2 | Runs F2.REL which is on DT1. |

## SUSPEND

**2.7.5.2** SUSPEND Command - The SUSPEND command is used to stop execution of the foreground job.

The form of the command ist

SUS{PEND}

No arguments are required. Foreground I/O transfers in progress will be allowed to complete; however, no new I/O requests will be issued and no completion routines will be entered (see Chapter 9 for a discussion of ccmpletion routines). Execution of the job can be resumed only from the keyboard.

**Example:**

. SUSPEND Suspends execution of the foreground job currently running.

**RSUME**

**2.7.5.3** RSUME Command - The **RSUME command** is used to resume execution of the foreground job where it was suspended. Any completion routines **which** were acheduled while the foreground was suapended are entered at this time.

The form of the command ist

RSU{ME}

No arguments are required.

Example:

```
. RSU      Resumes execution of the foreground job currently
           suspended.
```

## 2.8 MONITOR ERROR MESSAGES

The following error **messages** indicate fatal conditions that **can** occur **during system** boot:

| Message | Meaning |
|---|---|
| **?B-I/O ERROR** | An **I/O** error **occurred during system** boot. |
| **?B-NO** BOOT ON **VOLUME** | **No** bootstrap has been written on volume. |
| **?B-NO MONITR.SYS** | No monitor exists on volume being booted. |
| **?B-NOT** ENOUGB **CORE** | There is not enough **memory** for the **system** being booted **(e.g.,** attempting to **boot F/B** into **8K).** |

The following error **messages** are output by **the** Keyboard Monitor.

| Message | Meaning |
|---|---|
| **?ADDR?** | Address out of range in E or D command. |
| **?DAT?** | The DATE command **argument** was illegal, or ***no*** argument was given and the date **has** not yet been **set.** |
| **?ER** RD **OVLY?** | An **I/O** error occurred while **reading** a KMON overlay to process ***the*** **current** command. **This is** a serious error, indicating that **the system file MONITR.SYS** is unreadable. |
| *F?* | A CTRL F was typed **under** the **F/B** monitor andno foreground job exists. |
| **?F** ACTIVE? | Neither FRUN nor UNLOAD may be used when a foreground job already exists and **is active.** |
| **?FIL** NOT **FND?** | ***File*** specified in R, **RUN,** GET, or FRUN **command** not found. |
| ?FILE? | No file **named** where one ie expecteti. |

| Massage | Meaning |
|---|---|
| **?ILL CMD?** | Illegal Keyboard **Monitor command** *or* command line too long. |
| **?ILL** DEV? | Illegal or nonexistent device, or an attempt was made to make a device handler resident **for use** with a foreground job **(dev=F)** when the Single-Job Monitor was *running.* |
| ?NO CLOCIZ? | No **KW11L clock is** available for the TIME command. |
| ?NO FG? | A SUSPEND, **RSUME,** or **UNLOAD** FG command was given, but no foreground job was in memory. |
| **?OVR COR?** | Attempt to GET or RUN a file that is too big. |
| **?PARAMS?** | Bad Parameters were typed to the SAVE command. |
| **?REL** FIL **I/O** ER? | Either the **program** requested is not a **REL** file or a hardware *error was* encountered trying to read or write the **file.** |
| **?SV FIL I/O ER?** | **I/O** error on .SAV file in **SAVE** (output) or R, RUN, or GET **(input)** command. **Pos-**sible errors include end-of-file, hard error, and channel not open. |
| ?SY **I/O** ER? | **I/O** error on **system** device (usually reading **or** writing swap **area).** |
| **?TIM?** | The **TIME command** argument was illegal. |

**The** following messages are output by the RT-11 Resident Monitor when **an unre**coverable error has occurred. Control **passes** to the Keyboard **Monitor.** The **program** in **which** the error occurred **cannot** be restarted with the **RE command.** To execute the **program** again, **use** the **R** *or* RUN command.

The format **for** fatal monitor error messages ist

| | |
|---|---|
| **?M-text PC** | where PC ie the **address+2** of the location where the error occurred. |

Note that **?M** errors **can** be inhibited in certain **cases** by the use of the **.SERR macro;** see Chapter 9 **for details.**

| **Message** | **Meaning** |
|---|---|
| **?M-BAD FETCH** | **Either** an *error* occurred while reading 'in *a* device handler *from* SY, *or* the address at **which** the handler was to be loaded was illegal. |

| | |
|---|---|
| **?M-DIR** IO **ERR** | An error occurred doing **I/O** in the directory of a device **(e.g., .ENTER** on a **write-locked** device)'. |
| **?M-DIR** OVFLO | No **more** directory **segments** were available *for* expansion **(occurs during** file **creation (.ENTER)).** |
| **?M-DIR UNSAFE** | In **F/B** only, this message may appear in addition to any **of** the other diagnostics liated in this **section.** It indicates that **the** error occurred while the USR was updating a device directory. **One** *or* more files on that device may **be** lost. |
| **?M-FP** TRAP | *A* floating-point exception trap occutred, and the **user program** had no **.SFPA** exception routine **active (see** Chapter 9). |
| **?M-ILL** ADDR | **Under** the **F/B** Monitor, an addreea specified in a monitor **call** was odd or was not within the **job's** limits. |
| **?M-ILL CHAN** | **A** channel number was **specified** which was too large. |
| **?M-ILL EMT** | An **EMT** was executed which did not **exist;** i.e., the **function code** was out of bounds. |
| **?M-ILL** USR | The USR was **called** from a completion routine. **This** error does ***not*** have a *soft* return (i.e., **.SERR** will not inhibit **this message; see** Chapter 9). |
| **?M-NO** DEV | A **READ/WRITE** Operation was **tried** but no device handler was in **memory** for it. |
| **?M-OVLY ERR** | *A* **user program** with overlays failed to successfully read an overlay. |
| **?M-SWAP ERR** | A hard **I/O** error occurred while tho **system** was attempting to write a user **program** to the **system** swap **blocks.** This is usually **caused** by a **write-locked system** device. **Under** the **Single-**Job Monitor, this may **cause** the **system** to halt |
| ?M-SYS ERR | **An** I/O error occurred while trying to read **KMON/USR** into memory, indicating that the monitor file is situated on the **system** device in an area that has developed one or more bad **blocks.** The monitor prints the message and loops trying to read **KMON.** The message is a warning that the **system** device is bad. |

| | |
|---|---|
| | If, after several seconds, it is apparent that attempts to read KMON are failing, halt the processor. It may be impossible to boot the volume because of the bad area in the monitor file. Use another system device to verify the bad blocks and follow therecoveryprocedures described in section 4.2.12.1 of Chapter 4. |
| ?M-TRAP TO 4<br>?M-TRAP TO 10 | The job has referenced illegal memory or device registers, an illegal instruction was used, stack Overflow occurred, a word instruction was executed with an odd address, or a hardware problem caused bus time-out traps through location 4. |

If CSI errors occur and input was from the console terminal, an error message is printed on the terminal.

| Message | Meaning |
|---|---|
| ?DEV FUL? | Output file will not fit. |
| ?FIL NOT FND? | Input file was not found. |
| ?ILL CMD? | Syntax error. |
| ?ILL DEV? | Device specified does not exist. |

### 2.8.1 Monitor HALTS

There are two HALT instructions in the RT-11 V02 monitors, one each in F/B and Single-Job. The Single-Job Monitor will halt only if I/O errors occur during swap operations to the system device. If the S/J Monitor halts, look for a write-locked system device.

The F/B Monitor will halt if a trap to location 4 occurs or if I/O occurs while the system is perfonning critical operations from which it cannot recover. If the F/B Monitor halte, look for use of non-existent devices, traps from interrupt Service routines, or user-corrupted queue elements.

The monitor halts can be detected by their address, which is high in memory, above the resident base address (location 54).

When a monitor halt occurs, do not attempt to restart the system by pressing CONTinue on the processor; the system must be rebooted.

# CHAPTER 3

# TEXT EDITOR

The Text Editor (EDIT) ie used to create and modify ASCII source files so that these files can be used as input to other system programs such as the assembler or BASIC. Controlled by user commands from the keyboard, EDIT reads ASCII files from a storage device, makes specified changes and writes ASCII files to a storage device or lists them on the line printer or console terminal. EDIT allows efficient use of VT-11 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. Editing commands are then used to:

Locate text to be changed,

Execute and verify the changes,

Output a page of text to the output file,

List an edited page on the line printer or console terminal.

## 3.1 CALLING AND USING EDIT

To call EDIT from the system device type:

```
R EDIT
```

and the RETURN key in response to the dot (.) printed by the monitor. EDIT responds with an asterisk (*) indicating it is in command mode and awaiting a user command string.

Type CTRL C to halt the Editor at any time and return control to the monitor. To restart the Editor type .R EDIT or the .REENTER command in response to the monitor's dot. The contents of the buffers are lost when the Editor is restarted.

## 3.2 MODES OF OPERATION

Under normal usage, the Editor operates in one of two different modea: Command Mode or Text Mode. In Command Mode all input typed on the keyboard is interpreted as commands instructing the Editor to perform some Operation. In Text Mode all typed input is interpreted as text to replace, be inserted into, or be appended to the contents of the Text Buffer.

Immediately after being ioaded into memory and started, the Editor is in Command Mode. An asterisk is printed at the left margin of the console terminal page indicating that the Editor is waiting for the user to type a command. All commands are terminated by pressing the ALTMODE key twice in succession. Execution of commands proceeds from left to right. Should an error be encountered during execution of a command string, the Editor 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. The command in error (and any succeeding commands) is not executed and must be corrected and retyped.

Text mode is entered whenever the user types a command which must be followed by a text string. These commands insert, replace, exchange, or otherwise manipulate text; after such a command has been typed, all succeeding characters are considered part of the text string until an ALTMODE is typed. The ALTMODE terminates the text string and causes the Editor to reenter Command Mode, at which point all characters are considered commands again.

A special editing mode, called Immediate Mode, can be used whenever the VT-11 display hardware is running. This mode is described in Section 3.7.2.

## 3.3 SPECIAL KEY COMMANDS

The EDIT key commands are listed in Table 3-l. Control commands are typed by holding down the CTRL key while typing the appropriate Character.

Table 3-1
EDIT Key Commands

| Key | Explanation |
|---|---|
| ALTMODE | Echoes $. A single ALTMODE terminates a text string. A double ALTMODE executes the command string. For example,<br><br>*GMOV A, B$-1D$$ |
| CTBL C | Echoes at the terminal as ↑C and a carriage return. Terminates execution of EDIT commands, and returns to monitor Command Mode. A double CTRL C is necessary when I/O is in progress. The EEENTER command may be used to restart the Editor, but the contents of the text buffers are lost. |

(continued on next page)

Table 3-l **(cont.)**
EDIT Key Commands

| Key | Explanation |
|---|---|
| CTRLO | **Echoes** ↑O and a carriage return. Inhibits printing on the terminal **until** completion of the current **command** string. Typing a **second** CTRL 0 resumes output. |
| **CTRL u** | **Echoes** ↑U and a carriage return. Deletes all the **characters** on the current terminal input line. (Equivalent to typing RUBOUT back to the beginning of the **line.)** |
| RUBOUT | Deletes Character from the current line; **echoes** a backslash followed **by** the character deleted. **Each** succeeding RUBOUT typed by the **user** deletes and **echoes** another character. An enclosing backslash is printed when a key other than RUBOUT **is** typed. This erasure is done right to left up to the last carriage **return/line** feed combination. RUBOUT may be used -in both Command and Text Modes. |
| TAB | **Spaces** to the next tab stop. Tab **stops** are positioned every eight **spaces** on the terminal; typing 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 currently being entered. The Editor prints a <CR><LF> and an asterisk to indicate that the **user** may enter another command. For example:<br>`*IABCD`<br>`EFGH^X`<br>`*`<br>A CTRL U would only **cause** deletion of EFGH; CTRL X erases the entire command. |

## 3.4 COMMAND **STRUCTURE**

EDIT commands fall into six general categories:

| Category | **Commands** | **Section** |
|---|---|---|
| Input/Output | Edit Backup | 3.6.1.3 |
| | Edit Read | 3.6.1.1 |
| | Edit Write | 3.6.1.2 |
| | End File | 3.6.1.9 |
| | Exit | 3.6.1.10 |
| | List | 3.6.1.7 |
| | Next | 3.6.1.6 |
| | Read | 3.6.1.4 |
| | Verify | 3.6.1.8 |
| | Write | 3.6.1.5 |
| Pointer location | **Advance** | 3.6.2.3 |
| | Beginning | 3.6.2.1 |
| | **Jump** | 3.6.2.2 |

| | | |
|---|---|---|
| Search | Find | 3.6.3.2 |
| | Get | 3.6.3.1 |
| | Position | 3.6.3.3 |
| Text modification | Change | 3.6.4.4 |
| | Delete | 3.6.4.2 |
| | eXchange | 3.6.4.5 |
| | Insert | 3.6.4.1 |
| | Kill | 3.6.4.3 |
| Utility | Edit Console | 3.7.1 |
| | Edit Display | 3.7.1 |
| | Edit Lower | 3.6.5.6 |
| | Edit Upper | 3.6.5.6 |
| | Edit Version | 3.6.5.5 |
| | Execute Macro | 3.6.5.4 |
| | Macro | 3.6.5.3 |
| | Save | 3.6.5.1 |
| | Unsave | 3.6.5.2 |
| Immediate Mode | ALTMODE | 3.7.2 |
| | CTRL D | 3.7.2 |
| | CTRL G | 3.7.2 |
| | CTRL N | 3.7.2 |
| | CTRL V | 3.7.2 |
| | RUBOUT | 3.7.2 |

The general format for the first five categories of EDIT commands is:

```
nCtext$
```

or

```
nC$
```

where n represents one of the legal arguments listed in Table 3-2, C is a one- or two-letter command, and text is a string of successive ASCII characters.

As a rule, commands are separated from one another by a single ALTMODE; however, if the command requires no text, the separating ALTMODE is not necessary. Commands are terminated by a single ALTMODE; typing a second ALTMODE begins execution. (ALTMODE is used differently when Immediate Rode is in effect; Section 3.7.2 details its use in this case.)

The format of Display Editor commands is somewhat different from the normal editing command fonnat, and is described in Section 3.7.

### 3.4.1 Arguments

An argument is positioned before a command letter and is used either to specify the particular portion of text to be affected by the command or to indicate the number of times the command should be performe& with some commands, this specification is implicit and no arguments are needed; other editing commands require an argument. Table 3-2 lists the formats of arguments which are used by commands of this second type.

Table 3-2
**Command** Arguments

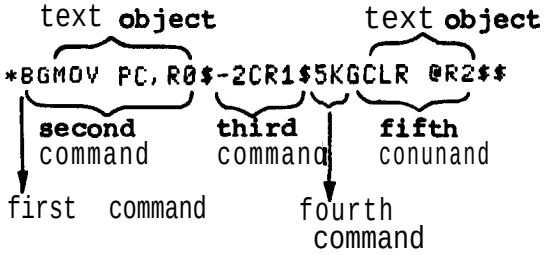
| Format | Meaning |
|---|---|
| n | n **stands for** any integer in the range -16383 to **+16383** and may, except where noted, be preceded by a + or **-.** ff no sign precedes n, it is assumed to be a positive number. Whenever an argument **is** acceptable in a **command, its** *absence* implies an argument of 1 (or -1 **if** only **the - is** present). |
| 0' | 0 refers to the beginning of the current line. |
| **/** | **/** refers **to** the end of **text** in the current Text Buffer. |
| **=** | **= is** used with the J, D and C **commands** *only* and represents **-n,** where n **is** equal to the length **of** the last text argurnent used. |

The roles of all arguments are explained more **specif** ically in following **sections.**

### *3.4.2* Command Strings

All EDIT command strings are terminated by two successive ALTMODE **characters. Spaces,** carriage returns and line feeds **within** a command string may be used freely to increase command readability but are ignored unless they appear *in* a taxt string. Commands used to **insert** text **can** contain text strings that are several lines long. **Each** line **is** terminated with a <CR><LF> and the entire command is terminated **with** a double ALTMODE.

Several commands **can** be strung together and executed in sequence. ***For*** example,




**Execution** of a command string begins when the double ALTMODE **is typed** and proceeds from left to right. Except when they are part of **a** text string, **spaces,** carriage return, line feed, and **single** ALTMODES are ignored. **For** example:

```
*BGMOV R0$=CCLR R1$AV$$
```

Table 3-2
Command Arguments

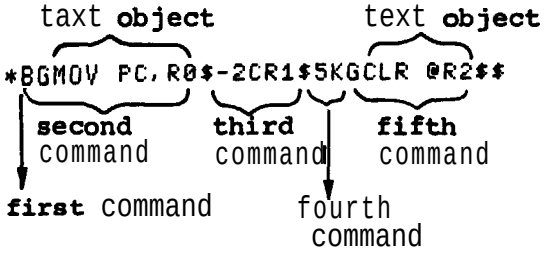
| Format | Meaning |
|---|---|
| n | n stands for any integer in the range -16383 to +16383 and may, except where noted, be preceded by a + or -. If no sign precedes n, it is assumed to be a positive number. Whenever an argument is acceptable in a command, fts absence implies an argument of 1 (or -1 if only the - is present). |
| 0' | 0 refers to the beginning of the current line. |
| / | / refers to the end of taxt in the current Text Buffer. |
| - | - is used with the J, D and C commands only and represents -n, where n is equal to the length of the last text argument used. |

The roles of all arguments are explained more specifically in following sections.

### 3.4.2 Command Strings

All EDIT command strings are terminated by two successive ALTMODE characters. Spaces, carriage returns and line feeds within a command string may be used freely to increase command readability but are ignored unless they appear in a text string. Commands used to insert text can contain text strings that are several lines lang. Each line is terminated with a <CR><LF> and the entire command is terminated with a double ALTMODE.

Several commands can be strung together and executed in sequence. For example,




Execution of a command string begins when the double ALTMODE ie typed and proceeds from left to right. Except when they are part of a text string, spaces, carriage return, line feed, and single ALTMODES are ignored. For example:

```
*BGMOV R0$=CCLR R1$AV$$
```

may be typed as:

```
*B$ GMOV R0$
=CCLR R1$
A$ V$$
```

with equivalent **execution.**

### 3.4.3 The Current Location Pointer

Most EDIT commands **function** with **respect** to a movable reference **pointer which** is normally located between the most **recent** Character operated upon and the next Character in the buffer. At any given time **during** the editing procedure, **this pointer can** be thought of as representing the current **position** of the Editor in the text. Most commands use **this pointer** as an **implied** argument. Commands are available for moving the **pointer** anywhere in the text, thereby redefining the current location and allowing greater facility in the use of other conunands.

### 3.4.4 **Character-** and Line-Oriented Command Properties

Edit commands are line-oriented or Character-oriented **depending** on the arguments they **accept. Line-oriented** commands operate on entire **lines** of text. Character-oriented commands operate on individual **characters** independent of what or where they are.

When using Character-oriented commands, a **numeric** argument **specifies** the number of **characters** that are involved in the Operation. Positive arguments represent the number of **characters** in a forward **direction** (in relation to the pointer), negative arguments the number of **characters** in a **backward direction.** Carriage return and line feed **characters** are treated the **same** as any other Character. For example, assume the **pointer** is positioned as indicated in the following text (t represents the current position of the pointer):

```
MOV    #VECT,R2<CR><LF>
CLR    @R2<CR><LF>
```

The EDIT command **-2J backs** the **pointer** by two **characters.**

```
MOV    #VECT,R2<CR><LF>
CLR    @R2<CR><LF>
```

The command **10J** advances the **pointer** forward by ten **characters** and **places** it between the CR and LF **characters** at the end of the **second** line.

```
MOV    #VECT,R2<CR><LF>
CLR    @R2<CR><LF>
```

Finally, **to place** the **pointer** after the **"C"** in the first line, a **-14J** command **is** used. The J (Jump) command **is** explained in **Section** 3.6.2.2.

```
MOV    #VECT,R2<CR><LF>
CLR    @R2<CR><LF>
```

When using line-oriented commands, a numeric argument represents the number of lines involved in the Operation. The Editor recognizes a line of text as a unit when it detects a <CR><LF> combination in the text. When the user types a carriage return, the Editor automatically inserts a line feed. Positive arguments represent the number of lines forward (in relation to the pointer); this is accomplished by counting carriage return/line feed combinations beginning at the pointer. So, if .the pointer is at the beginning of a line, a line-oriented conunand argument of +1 represents the entire line between the current pointer and the terminating line feed. If the current pointer is in the middle of the line, an argument of +1 represents only the portion of the line between the pointer and the terminating line feed.

For example, assume a buffer of:

```
MOV    ↑PC,R1<CR><LF>
ADD    #DRIV-.,R1<CR><LF>
MOV    #VECT,R2<CR><LF>
CLR    @R2<CR><LF>
```

The command to advance the pointer one line (1A) causes the following change:

```
MOV    PC,R1<CR><LF>
↑ADD   #DRIV-.,R1<CR><LF>
MOV    #VECT,R2<CR><LF>
CLR    @R2<CR><LF>
```

The command 2A moves the pointer over 2 <CR><LF> combinations:

```
MOV    PC,R1<CR><LF>
ADD    #DRIV-.,R1<CR><LF>
MOV    #VECT,R2<CR><LF>
↑CLR   @R2<CR><LF>
```

Negative line arguments reference lines in a backward direction (in relation to the pointer). Consequently, if the pointer is at the beginning of the line, a line argument of -1 means "the previous line" (moving backward past the first <CR><LF> and up to but not including the second <CR><LF>; if the printer is in the middle of a line, an argument of -1 means the preceding 1 1/2 lines. Assume the buffer contains:

```
MOV    PC,R1<CR><LF>
ADD    #DRIV-.,R1<CR><LF>
MOV    #VECT,R2<CR><LF>
CLR    @R2<CR><LF>
```

A command of -1A backs the pointer by 1 1/2 lines.

```
MOV    PC,R1<CR><LF>
↑ADD   #DRIV-.,R1<CR><LF>
MOV    #VECT,R2<CR><LF>
CLR    @R2<CR><LF>
```

Now a command of **-1A backs** it by only 1 line.

```
↑MOV    PC,R1<CR><LF>
 ADD    #DRIV-.,R1<CR><LF>
 MOV    #VECT,R2<CR><LF>
 CLR    @R2<CR><LF>
```

### 3.4.5 Command Repetition

**Portions** of a command string may be executed more than once by enclosing the desired portion in angle brackets **(<>)** and preceding the left angle bracket with the number **of** iterations desired. The structure is:

```
C1$C2$n<C3$C4$>C5$$
```

where C1, C2 **,...C5** represent commands and n represents an iteration argument. Commands C1 and **C2** are **each** executed once, then commands C3 and **C4** are executed n times. Finally command **C5 is** executed once and the command line **is** finished. The iteration argument **(n)** must be a positive number (1 to **16,383),** and **if** not specified is assumed to be 1. If the number **is** negative or too large, an error message is printed. Iteration brackets may be nested up to 20 levels. Command **lines** are **checked** 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$$
```

is equivalent to typing:

```
*BGAAA$-DIB$-J-DIB$-J-DIB$-JV$$
```

and:

```
*B3<2<AD>V>$$
```

**is** equivalent to typingt

```
*BADADVADADVADADV$$
```

**The** following bracket structures are examples of legal usager

```
<<><<<><>>>>
<<<>>><><>
```

**The following** bracket structures are examples **of** illegal combinations **which** will **cause** an error message **since** the brackets are not properly matchedt

```
><><
<<<>>
```

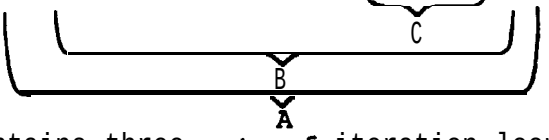
**During** command repetition, execution proceeds from left to right until a right bracket **is** encountered. EDIT then returns to the last left

bracket encountered, decrements the iteration counter and executes the commands within the brackets. When the counter is decremented 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 innennost brackets and then works its way back again. The most common use for iteration brackets is found in commands such as Unsave, that do not accept repeat counts. For examplet

```
*3<U>$$
```

Assume a file called SAMP (stored on device DK) is to be read and the first four occurrences of the instruction MOV #200,R0 on each of the first five pages are to be changed to MOV #244,R4. The following command line is entered:

```
*EBSAMP$5<N4<BGMOV #200,R0$=J$3<G0$=C4$>>>EX$$
```

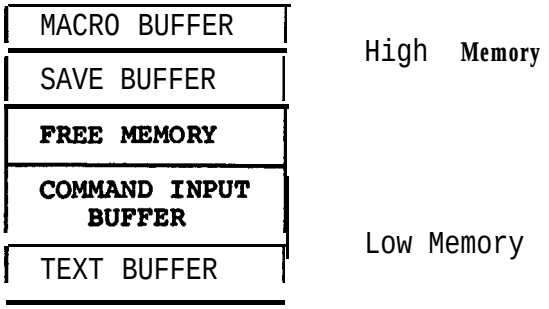



The command line contains three sets of iteration loops (A,B,C) and is executed as follows:

Execution inftially proceeds from left to right; the file SAMP is opened for input, and the first page is read into memory. The pointer is moved to the beginning of the buffer and a search is initiated for the Character string MOV #200,R0. When the string is found, the pointer is positioned 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 4 times, control moves back to the second iteration of loop A, and so forth until all iterations have been satisfied.

## 3.5 MEMORY USAGE

The memory area used by the Editor is divided into four logical buffers as follows:



The Text Buffer contains the curtent page of text being edited, and the Command Input Buffer holds the command currently being typed at the terminal. If a command currently being entered by the user is within 10 characters of exceeding the space available in the Command Buffer, the message:

```
* CB ALMOST FULL *
```

is printed. If the command can be completed within 10 characters, the user may finish entering the command; otherwise he should type the ALTMODE key twice to execute that portion of the command line already completed. The message is printed each time a character is entered in one of the last 10 spaces.

If the user attempts to enter more than 10 characters the message:

```
?CB FULL?
```

is printed and all conunands typed within the last 10 characters are ignored. The user again has 10 characters of available space in which to correct the condition.

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 3.6.5.)

The Macro and Save Buffers are not allocated space until an M or S command is executed. Once an M or S command is executed, a OM or OU (Unsave) command must be executed to return that space to the free area.

The size of each buffer automatically expands and contracts to accommodate the text being entered; if there is not enough space available to accommodate required expansion of any of the buffers, a "?*NO ROOM*?" error message is typed.

## 3.6 EDITING COMMANDS

### 3.6.1 Input/Output Commands

Input commands are used to create files and read them into the Text Buffer where they become available for editing or listing. output commands cause text to be listed on the console terminal or line-printer or written out to a storage device. Some commands are specifically designed for either input or output functions, while a few commands serve both purposes.

Once editing is completed and the page currently in the Text Buffet is written to the output file, that page of text is unavailable for further editing until the file is closed and reopened.

3.6.1.1 Edit Read - The Edit Read command opens an existing file for input and prepares it for editing. Only one file can be open for input at a time.

The form of the command is:

ERdev:filnam.ext$

The string argument (dev:filnam.ext) is limited to 19 characters and specifies the file to be opened. If no device is specified, DK: is assumed. If a file is currently open for input, that file is closed; any edits made to the file are preserved.

Edit Read does not input a page of text nor does it affect the contents of the other user buffers (see Section 3.5.)

Edit Read can be used on a file which 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.

Examples:

| | |
|---|---|
| *ERDT1:SAMP.MAC$$ | Opens SAMP.MAC on device DT1: for input. |
| *ERSOURCE$$ | Opens SOURCE on device DK: for input. |

3.6.1.2 Edit Write - The Edit Write command sets up a file for output of newly created or edited text. Bowever, no text is output and the contents of the user buffers are not affected. Only one file can be open for output at a time. Any current output files are closed

The form of the command is:

EWdev:filnam.ext[n]$

The string argument (dev:filnam.ext[n]) is limited to 19 characters and is the name to be assigned to the output file being opened. If devt is not specified, DK: is assumed. [n] is optional and represents the length of the file to be opened. If not specified, one half the largest available space is used; if this is not adequate for the output file size, the EF and EX commands will not close the output file, and all edits will be lost. It is thus recommended that the [n] construction be used whenever there is doubt as to whether enough space is available on the device for the output file.

If a file with the same name already exists on the device, the old file is deleted when an EXit, End File or another Edit Write command is executed.

Examples:

| | |
|---|---|
| *EWDK:TEST.MAC$$ | Opens the file TEST.MAC on device DK: for output. |
| *EWFILE.BAS[11]$$ | Opens the file FILE.BAS (allocating 11 blocks) on the device DK: for output. |

3.6.1.3 Edit Backup - The Edit Backup command is used to open an existing file for editing and at the same time create a backup Version of the file. Any currently open file will be closed. No text is read or written with this command.

The form of the command ist

**EBdev:filnam.ext[n]$**

The device designation, filename and extension are limited to 19 **characters.** If dev: **is** not specified, **DK: is** assumed. **[n] is** optional and represents the length of the file to be opened; **if** not specified, one-half the largest available **space** is used.

The file indicated in the command line muat already exist on the device **designated since** text will be read from **this** file as input. At the **same time,** an output file **is** opened **under** the **same** filename and extension. After an EB command has been successfully executed, the original file **(used** as input) **is** renamed with the current filename and a .BAX extension; any previous file with **this** filename and a .BAX extension **is** deleted. **The** new output file **is** closed and assigned the name as specified in the EB command. This renaming of files takes **place** whenever an **Exit,** End File, Edit Read, Edit Write or Edit Backup command **is** executed.

**Examples:**

| | |
|---|---|
| *EBSY: BAS1.MAC$$ | Opens **BAS1.MAC** on SY. When editing is complete, the old **BAS1.MAC** becomes BAS1.BAX and the new file becomes **BAS1.MAC.** Any previous Version of BAS1.BAX **is** deleted. |
| *EBBAS2.BAS[15]$$ | **Opens BAS2.BAS** on DK **(allocating** 15 **blocks).** When editing **is** complete, **the** old **BAS2.BAS** is labeled **BAS2.BAK** and the new file becomes **BAS2.BAS,** Any previous Version of **BAS2.BAK is** deleted. |

In **EB,** ER and EW **commands, leading spaces** between the command and the filename are illegal **(the** filename is considered to be a text string). All **dev:file.ext** specifications for EB, ER and EW commands **conform** to the RT-11 **conventions** for file naming and are identical to filenames entered in command strings **used** with **other system** programs.

3.6.1.4 Read - The Read ccmmand **(R) causes** a page of text to be read from the input file **(previously** specified in an ER or EB command) and appended to the current contents, **if** any, of the Text Buffer.

The form of the command **is:**

R

No arguments are used **with** the R command and the **pointer is** not moved. Text **is** input **until** one of **the** following conditions **is** met:

1. **Aform** feed Character, signifying the end of the **page, is encountered.** At this **point,** the form feed will be the last Character in the buffer; or

2. The Text Buffer is within 500 characters of being full. (When this condition occurs, Read inputs up to the next <CR><LF> combination, then returns to Command Mode. An asterisk ie printed as though the Read were complete, but text will not have been fully input); or

3. An end-of-file condition is detected, (the *EOF* message is printed when all text in the file has been read into memory and no more input is available).

The maximum number of characters which can be brought into memory with an R command is approximately 6,000 for an 8K System. Each additional 4K of memory allows approximately 8,000 additional characters to be input. An error message is prfnted if the Read exceeds the memory available or if no input is available.

3.6.1.5 Write - The Write command (W) moves lines of text from the Text Buffer to the output file (as specified in the EW or EB command). The format of the command is:

**nW** Write all characters beginning at the pointer and endjng at the nth <CR><LF> to the output file.

**-nW** Write all characters beginning on the -nth line and terminating at the pointer to the output file.

**0W** Write the text from the beginning of the current line to the pointer.

**/W** Write the text from the pointer to the end of the buffer.

The pointer is not moved and the contents of the buffer are not affected. If the buffer is empty when the Write is executed, no characters are output.

Examples:

| | |
|---|---|
| *5W$$ | Writes the next 5 lines of text starting at the pointer, to the current output file. |
| *-2W$$ | Writes the previous 2 lines of text, ending at the pointer, to the current output file. |
| *B/W$$ | Writes the entire Text Buffer to the current output file. |

3.6.1.6 Next - The Next command acts as both an input and output command since it performs both functions. First it writes the current Text Buffer to the output file, then clears the buffer, and finally reads in the next page of the input file. The Next command can be repeated n times by specifying an argument before the command. The command format is:

nN

Next accepts only positive arguments (n) and leaves the pointer at the beginning of the buffer. If fewer than n pages are available in the input file, all available pages are input to the buffer, output to the current file, and deleted from the buffer; the pointer is left positioned at the beginning of an empty buffer, and an error message is printed. This command is equivalent to a combination of the Beginning, Write, Delete and Read commands (B/W/DR). Next can be used to space forward, in page increments, through the input file.

Example:

| | |
|---|---|
| *2N$$ | Writes the contents of the current Text Buffer to the output file. Read and write the next page of text. Clear the buffer and then read in another page. |

3.6.1.7 List - The List command prints the specified number of lines on the console terminal. The format of the command is:

| | |
|---|---|
| nL | Print all characters beginning at the pointer and ending with the nth <CR><LF>. |
| -nL | Print all characters beginning with the first Character on the -nth line and terminating at the pointer. |
| 0L | Print from the beginning of the current line up to the pointer. |
| /L | Print from the pointer to the end of the buffer. |

The pointer is not moved after the command is executed.

Examples:

| | |
|---|---|
| *-2L$$ | Prints all characters starting at the second preceding line and ending at the pointer. |
| *4L$$ | Prints all characters beginning at the pointer and terminating at the 4th <CR><LF>. |

Assuming the pointer location is:

```
MOVB 5(R1),@R2
ADD  R1,(R2)+
```

The command:

```
*-1L$$
```

Prints the previous 1 1/2 lines up to the pointer:

```
MOVB 5(R1),@R2
ADD
```

3.6.1.8 Verify - The Verify command prints the current text line (the line containing the pointer) on the terminal. The position of the pofnter within the line has no effect and the pointer does not move . The command format is:

```
V
```

No arguments are used. The V command is equivalent to a OLL (List) command.

**Example:**

```
*V$$
ADD     R1,(R2)+
```

The command causes the current line of text to be printed.

3.6.1.9 End File - The End File command closes the current output file. This command does no input/output operations and does not move the pointer. The buffer contents are not affected. The output file is closed, containing only the text previously output.

The form of the command is:

```
EF
```

No arguments are used. Note that an implied EF command is included in EW and EB commands.

3.6.1.10 EXit - The EXit command is used to terminate editing, copy the text buffer and the remainder of the input file to the output file, close input and output files, and return control to the monitor. It performs consecutive Next cormnands until the end of the input file is reached, then closes both the input and output files.

The command fonnat is:

```
EX
```

No arguments are used. Essentially, Exit is used to copy the remainder of the input file into the output file and return to the monitor. Exit is legal only when there is an output file open. If an output file is not open and it is desired to terminate the editing Session, return to the monitor with CTRL C.

NOTE

An EF or EX command is necessary in order to make an output file permanent. If CTRL C is used to return to the monitor without a prior execution of an EF command, the current output file ie not saved. (It can however, be made permanent using the monitor CLOSE command; see Section 2.7.2.5.)

An example of the contrasting uses of the EF and EX commands follows. Assume an input file, SAMPLE, contains several pages of text. The user wishes to make the first and second pages of the file 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$$
*EWSAMPLE$EX$$
```

The user might note that the EF commands are not necessary in this example since the EW Command closes a currently open output file before opening another.

### 3.6.2 Pointer Relocation Commands

Pointer relocation commands allow the current location pointer to be moved within the Text Buffer.

3.6.2.1 Beginning - The Beginning command moves the current location pointer to the beginning of the Text Buffer.

The command format 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:

```
*B$$
```

moves the **pointer** to the beginning of the Text Buffer:

```
MOVB    5(R1),@R2
ADD     R1,(R2)+
CLR     @R2
MOVB    6(R1),@R2
```

3.6.2.2 Jump - The Jump command moves the **pointer** over the specified number of characters in the Text Buffer.

The form of the command is:

| | |
|---|---|
| **(+ or -) nJ** | Move the **pointer** (backward **or forward)** n characters. |
| 0J | Move the **pointer** to the beginning of the **current line (equivalent** to 0A). |
| **/J** | Move the **pointer** to the end of the Text Buffer **(equivalent to /A).** |
| **=J** | Move 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 toward the end. Jump treats carriage return, line feed, and form feed **characters** the same as any other **character,** counting one buffer position for **each.**

Bxamples:

| | |
|---|---|
| *3J$$ | Moves the **pointer** ahead three characters. |
| *-4J$$ | Moves the **pointer** back four characters. |
| *B$$GABC$=J$$ | Move the **pointer** so that it immediately precedes the first occurrence of **'ABC'** in the buffer. |

3.6.2.3 **Advance** - The **Advance** command **is** similar to the Jump command except that it moves the **pointer** a specified number of lines (rather than **single** characters) and leaves it positioned at the beginning of the line.

The form of the command is:

| | |
|---|---|
| **nA** | **Advance** the **pointer** forward n lines and position it at the beginning of the nth line. |

| | |
|---|---|
| **nA** | Move the **pointer backward** past n **<CR><LF> combinations** and position it at the beginning of the **-nth line.** |
| 0A | **Advance** the **pointer** to the beginning of the current line (equivalent to **0J).** |
| **/A** | **Advance** the **pointer** to the end of the Text Buffer (equivalent to **/J).** |

Examples:

| | |
|---|---|
| ***3A$$** | **Moves** the **pointer** ahead three lines. |

Assuming the buffer contains:

```
CLR     @R2
```

The command:

```
*0A$$
```

Moves the **pointer** to:

```
↑CLR    @R2
```

### 3.6.3 Search Commands

Search commands are used to **locate** specific characters or strings of characters within the Text Buffer.

3.6.3.1 Get - The Get command **starts** at the **pointer** and searches the current Text Buffer for the nth occurrence of a specified text string. If the search **is** successful, the **pointer is** left immediately following the nth occurrence of the text string. If the search fails, an error message is printed and the **pointer** is left at the end of the Text Buffer. The format of the command is:

```
nGtext$
```

**The** argument **(n)** must be positive and **is** assumed to be 1 **if** not otherwise specified. **The** text string may be any length and immediately follows the G command. The search is made on the portion of the text between the **pointer** and the end of the buffer.

Example:

Assuming the buffer **contains:**

```
↑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 command:

```
*GADD$$
```

positions the pointer at:

```
ADD.    #DRIV-.,R1
```

The commandt

```
*3G@R2$$
```

positions the pointer at:

```
ADD     R1,(R2)+
CLR     @R2,
```

After search commands, the pointer is left immediately following the text object. Using a search command in combination with =J will place the pointer before the text Object, as follows:

```
*GTEST$=J$$
```

This command combination places the pointer before 'TEST'.

3.6.3.2 Find - The Find command starts at the current pointer and searches the entire input file for the nth occurrence of the text string. If the nth occurrence of the text string is not found in the current buffer, a Next command is automatically performed and the search is continued on the new text in the buffer. When the search is successful, the pointer is left immediately following the nth occurrence of the text string. ff the search fails (i.e., the end-of-file is detected for the input file and the nth occurrence of the text string has not been found), an error message is printed and the pointer is left at the beginning of an empty Text Buffer.

The form of the command is:

```
nFtext$
```

The argument (n) muet be positive and is assumed to be 1 if not otherwise specified.

By deliberately specifying a nonexistent search string, the user can close out his file; that is, he can copy all remaining text from the input file to the output file.

Find is a combination of the Get and Next commands.

Examplet

```
*2FMOVB 6(R1),@R2$$
```

Searches the entire input file for the second occurrence of the text string MOVB 6(R1),@R2. Each unsuccessfully searched buffer is written to the output file.

3.6.3.3 Position - The Position command searches the input file for the nth occurrence of the text string. If the desired text string is not found in the current buffer, the buffer is cleared and a new page is read from the input file. The fonnat of the command is:

**nPtext$**

The **argument (n)** must be positive, and is assumed to be 1 if not otherwise specified. When a P command is executed the current contents of the buffer are searched from the location of the **pointer** to the end of the buffer. If the search is **unsuccessful,** the buffer is cleared and a new page of text is read and the cycle is continued.

If the search is successful, the **pointer is** positioned after the nth occurrence of the text. If it is not, the **pointer is** left at the beginning of an empty Text Buffer.

The Position **command is** a combination of the Get, Delete and **Read** connnands; it is most useful as a means of placing the location **pointer** in the input file. For example, if the aim of the editing Session is to **create** a new file from the **second** half of the input file, a Position search will save **time.**

The **difference** between the Find and Position commands is that Find **writes** the contents of the searched buffer to **the** output file while Position deletes the contents of the buffer after it is searched.

**Example:**

| | |
|---|---|
| *PADD R1,(R2)+$$ | Searches the entire input file for the specified string ignoring the unsuccessfully searched buffers. |

3.6.4 Text **Modification** Commands

**The** following **commands** are used to insert, relocate, and delete text in the Text Buffer.

3.6.4.1 Insert - The Insert command **causes** the Editor to enter Text Mode and allows text to be inserted immediately following the **pointer.** Text is inserted **until** an ALTMODE is typed and the **pointer is** positioned **immediately** after the last Character of the insert. **The command** format ist

**Itext$**

No arguments are used 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 ALTMODE are legal in the text string. ALTMODE **terminates** the text string.

NOTE

Forgetting to type the 1 command will **cause** the text **entered** to be executed as connnands.

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 is printed.

Example:

```
*IMOV   #BUFF,R2
MOV     #LINE,R1
MOVB    -1(R2),R0$$
*
```

Inserts the specified text at the current location of the pointer and leaves the pointer positioned after R0.

3.6.4.2 Delete - The Delete command removes a specified number of characters from the Text Buffer. Characters are deleted starting at the pointer. Upon completion, the pointer is positioned at the first character following the deleted text.

The form of the command is:

| | |
|---|---|
| (+ or -) nD | Delete n characters (forward or backward from the pointer). |
| 0D | Delete from beginning of current line to the pointer (equivalent to 0K). |
| /D | Delete from pointer to end of Text Buffer (equivalent to /K). |
| =D | Delete -n characters, where n equals the length of the last text argument used. |

Examples:

| | |
|---|---|
| *-2D$$ | Deletes the two characters immediately preceding the pointer. |
| *B$FMOV R1$=D$ | Deletes the text string 'MOV R1'. (=D used in combination with a search command will delete the indicated text string). |

Assuming a buffer of:

```
ADD     R1,(R2)+
CLR     @R2
```

the command:

```
*0D$$
```

leaves the buffer with:

```
ADD     R1,(R2)+
@R2
```

3.6.4.3 Kill - The Kill command removes n lines from the Text Buffer. Lines are deleted starting at the location pointer; upon completion of the command, the pointer is positioned at the beginning of the line following the deleted text. The command format ist

| | |
|---|---|
| nK | Delete lines beginning at the pointer and ending at the nth <CR><LF>. |
| -nK | Delete lines beginning with the first Character in the -nth line and ending at the pointer. |
| 0K | Delete from the beginning of the current line to the pointer (equivalent to 0D). |
| /K | Delete from the pointer to the end of the Text Buffer (equivalent to /D). |

**Example:**

| | |
|---|---|
| *2K$$ | Delete lines starting at the current location pointer and ending at the 2nd <CR><LF>. |

Assuming a buffer oft

```
ADD     R1,(R2)+
CLR     @R2
MOVB    6(R1),@R2
```

the command:

```
*/K$$
```

alters the contents of the buffer to:

```
ADD     R1,(R2)+
CLR
```

Kill and Delete commands perform the same function, except that Kill is line-oriented and Delete is Character-oriented.

3.6.4.4 Change - The Change connnand replaces n characters, starting at the pointer, with the specified text string and leaves the pointer positioned immediately following the changed taxt.

The form of the command is:

| | |
|---|---|
| (+ or -) nCtext$ | Replace n characters (forward or backward from the pointer) with the specified text. |
| 0Ctext$ | Replace the characters from the beginning of the line up to the pointer with the specified text (equivalent to 0x). |
| /Ctext$ | Replace the characters from the pointer to the end of the buffer with the specified text (equivalent to /X). |

**=Ctext$** Replace **-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 ALTMODE **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 so that the command **can** 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 **is** printed. The Change **command is** identical to executing a **Delete** command followed by an Insert **(nDItext$).**

Examples:

***5C#VECT$$** Replaces the five characters to the right of the **pointer** with **#VECT.**

Assuming a buffer oft

```
CLR     @R2
MOV+    5(Rl),@R2
```

The command:

***0CADDB$$**

leaves the buffer with:

```
CLR     @R2
ADDB↑   5(Rl),@R2
```

**=C can** be used in conjunction with a search command to replace a specific text string **as** follows:

***GFIFTY:$=CFIVE:$** Find the occurrence of the text string FIFTY: and replace it with the text string FIVE:.

3.6.4.5 Exchange - The Exchange command **exchanges** n lines, beginning at the **pointer,** with the indicated text string and leaves the **pointer** positioned after the **changed** text.

The form of the command **is:**

**nXtext$** Exchange all characters beginning at the **pointer** and ending at the nth <CR><LF> with the indicated text.

**-nXtext$** Exchange all characters beginning with the first Character on the **-nth** line and ending at the **pointer** with the indicated text.

**0Xtext$** Exchange the **current** line from the beginning to the **pointer** with the specified text (equivalent to **0C).**

/Xtext$ Exchange the *lines* from the **pointer** to the end of the buffer with the specffed text **(equivalent** to **/C).**

**All characters** are legal in the text string except ALTMODE **which** terminates the text.

The Exchange **command is** identical to a Kill command followed by an Insert **(nKItext$),** and accepts all legal **line-oriented** arguments.

If the X command is enclosed **within** angle **brackets** so that it will be executed more than once, and if there is enough memory **space** available so that the X command **can** be entered, it will be executed at least once (provided it **is** first *in* the command string). If repetition of the command exceeds the available memory, *an error* message **is** printed.

**Example** :

```
*2XADD  R1, (R2)+
CLR     @R2
$$
*
```

Exchange8 the two lines to the right of the **pointer** location with the text string.

### 3.6.5 Utility Commands

3.6.5.1 Save - The Save **command starts** at the **pointer** and copies the specified number of lines into the Save Buffer (described previously in **Section** 3.5).

The form **of** the command **is:**

**nS**

The argument **(n)** must be positive. The **pointer** position does not **change** and the contents of the Text Buffer are not altered. **Each** time a Save **is** executed, the previous contents of the Save Buffer, if any, are destroyed. rf the Save command **causes** an Overflow of the Save Buffer, an error message **is** printed.

Examplet

Assume the Text Buffer contains *the following* assembly language subroutiner

```
;SUBROUTINE   MSGTYP
;WHEN CALLED, EXPECTS RO TO POINT TO AN
;ASCII MESSAGE TRAT ENDS IN A ZERO BYTE,
;TYPES TRAT MESSAGE ON THE USER TERMINAL

              .ASECT
              .=1000
MSGTYP:       TSTB (%0)          ;DONE?
              BEQ MDONE          ;YES-RETURN
MLOOP:        TSTB @#177564      ;NO-IS TERMINAL RBADY?
              BPL MLOOP          ;NO-WAIT
              MOVB(%0)+,@#177566 ;YES PRINT CHARACTER
              BR MSGTYP          ;LOOP
MDONE:        RTS %7             ;RETURN
```

The command:

```
*145$$
```

stores the entire subroutine in the Save Buffer; it may then be inserted in a **program** wherever needed by using the U command.

3.6.5.2 **Unsave** - The **Unsave** command inserts the entire contents of the Save Buffer into the Text Buffer at the **pointer** location and leaves the **pointer** positioned following the inserted text.

The form of the command ist

| | |
|---|---|
| U | Insert in the Text Buffer the contents of the Save Buffer. |
| **OU** | Clear the Save Buffer and reclaim the area for text. |

Zero is the only legal argument to the U **command.**

The contents of the Save Buffer are not destroyed by the **Unsave** command (only by the OU command) and may be Unsaved as many times as desired.

If there is no text in the Save Buffer and the U command is given, the ?*NO TEXT*? error message is printed. If the **Unsave** command **causes** an Overflow of the Text Buffer, the ?*NO **ROOM*?** error **message is** displayed.

3.6.5.3 Macro - The Macro command inserts a c**ommand** string into the EDIT Macro Buffer. The Macro command **is** of the form:

| | |
|---|---|
| **M/command string/** | Store the command string in the Macro Buffer. |
| OM | Clear **the** Macro Buffer and reclaim the |
| or | area for text. |
| **M//** | |

/ represents the delimiter Character. The delimiter **is** always the first Character following the M command, and may be any Character **which** 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 it can be executed later by the Execute Macro (EM) command. Macro does not affect the contents of the Text or Save Buffers.

All characters except the delimiter are legal Macro command string characters, including single ALTMODE8 to terminate text commands. All commands, except the M and EM commands, are legal in a command string macro.

In addition to the OM command, typing the M command immediately followed by two identical characters (assumed to be delimiters) and two ALTMODE characters also clears the Macro Buffer.

Examples:

| | |
|---|---|
| *M//$$ | Clears the Macro Buffer |
| *M/GR0$-C1$/$$ | 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$=CADD R1$ $$
?*NO FILE*?
```

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 the R0$= becomes an illegal command structure.

3.6.5.4 Execute Macro - The Execute Macro command executes the command string specified in the last Macro command.

The form 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.

Examples:

```
*M/BGR0$-C1$/$$
*B1000EM$$
?*SRCH FAIL I  N   MACRO*?
*
```

Executes the **MACRO** stored in the previous example. **An** *error* message **is** returned when the end of buffer **is** reached. **(This macro** effectivelychanges all occurrences **of** RO in the Text Buffer **to Rl.)**

```
*IMOV PC,R1$2EMICLR @R2$$
*
```

In a new **program,** inserts MOV **PC,Rl** then executes the command in **the Macro** Buffer twice before inserting CLR **@R2.**

3.6.5.5 Edit Version - The Edit Version **command** displays the Version **number** of the Editor in use on the **console** terminal.

The form of the command ie:

**EV$**

Example:

```
*EV$$
V02-01
*
```

3.6.5.6 Upper- and Lower-Case Commands - **Users** who have any **upper/** lower-case terminal as part of their hardware **configuration may** 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 called **(R** EDIT), upper-case mode is **assumed;** all **characters** typed are automatically translated to upper **case. To** allow processing of both upper- and lower-case **characters,** the Edit **Lower** command is entered:

```
*EL$$
*i Text and commands can be entered in UPPER and lower case.$$
*
```

The Editor now accepts and **echoes** upper- and lower-case **characters** received from the keyboard, and Outputs text on the teleprinter in upper- and lower-case.

To return to upper-case mode, the Edit Upper command is used:

***EU$$**

Control also reverts **to** upper-case mode upon exit from the **Editor (via** EF, EX, or CRTL C).

Note that when an EL command has been issued, Edit commands can be entered in either upper- or lower-case. Thus, the following twocommands 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.

## 3.7 THE DISPLAY EDITOR

In addition to all **functions** and commands mentioned thus far, the Editor has additional capabilities to allow **efficient** use **of** VP-11 display hardware **which** may be part of the **system configuration (GT40,** GT44, DECLAB **11/40).**

The most apparent feature **is** the ability to use the display **screen** rather than the **console** terminal as a window into the Text Buffer for printout of all textual input and output. When all the features of the display Editor are in use, a 12" **screen** displays text as shown in Figure 3-l:

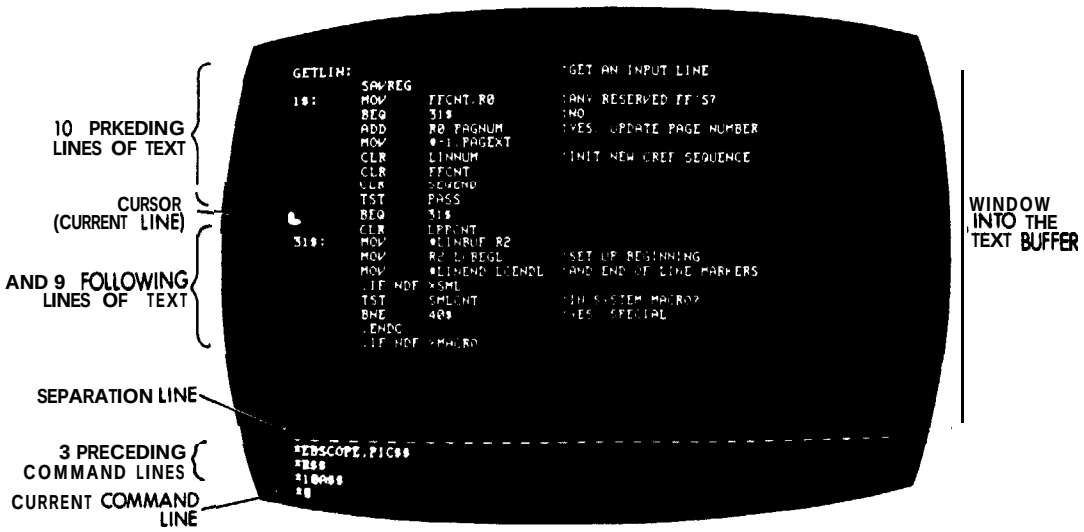



Figure 3-l
Display Editor Format

The major advantage is that the **user can now** see immediately where the **pointer** is. The **pointer** appears between **characters** on the **screen** as a bright blinking L-shaped cursor and **can** be detected **easily** and quickly. 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 10 lines of text preceding the current line and the 9 lines following it are also in view. **Each time** a command string is executed (via a double ALTMODE) this portion of the **screen** is refreshed so that it reflects the results of the **commands** just performed.

The lower eection of the **screen** contains 4 lines of editing commands. The command line currently being entered is last, preceded by the three most **recent** command lines. **This section is** separated from the text portion of the **screen** by a horizontal line of dashes. As new command lines are entered, previous command lines are scrolled upward off the command **section** so that only four command lines are ever in view.

A 17" **screen** displays 30 lines of text and 8 command lines.

### 3.7.1 Using the Display Editor

The display features of the Editor are automatically invoked whenever the **system** scroller is in use and the user **types:**

```
R EDIT
```

Hcwever, if the **system** does not contain **VT-11** display hardware, the display features are not enabled.

Providing that the **system** does contain **VT-11** display hardware and that the user wishes to employ the **screen during** the editing Session, he may activate it in one of two ways (all editing commands and **functions** previously discussed in this **chapter** are valid for use):

1. ff the scroller is in use (i.e., the GT ON monitor command has been typed **prior** to calling the Editor), EDIT recognizes this and automatically continues using 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 is used to display command lines. This arrangement is shown in Figure 3-l.

   The Edit Console command **can** be used to return the scroller to **its** normal mode so that text and comrnands appear as described in Chapter 2, **Section** 2.7.1 (i.e., using the full **screen for** display of command lines, and eliminating the **window).** The **form** of the command is:

   ```
   EC
   ```

For example:

*BAEC2L$$ — The second and third lines of the current buffer are listed on the Screen; there is no window into the Text Buffer at this point.

Subsequent EC commands are ignored if the window into the Text Buffer is not being displayed.

To recall the window, the Edit Display command is used:

ED

The screen is again arranged as shown in Figure 3-l.

2. Assume the scroller is not in use (i.e., the GT ON command has not been typed, or the monitor GT OFF command has been typed prior to calling the Editor). When the user calls EDIT, an asterisk appears on the console terminal as described in Section 3.1. Using the ED command at this time provides the window into the Text Buffer; however, commands continue to be echoed to the console terminal.

   When ED is used in this case, it must be the first command issued; 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). The dfsplay cannot be used again until a fresh copy of EDIT is loaded.

   While the display of the text window is active, ED commands are ignored.

   Typing the EC command clears the screen and returns all output to the console terminal.

NOTE

Under the Single-Job Monitor only, after the editing Session is over, it is recommended that the screen be cleared by either typing the EC command, *or* returning to the monitor and using the monitor INITIALIZE command. Failure to do this may cause unpredictable results.

### 3.7.2 Setting the Editor to Immediate *Mode*

An additional mode is available in EDIT to provide an easier and faster degree of interaction during the editing session. This mode is called Immediate Mode and combines the most-used functions of the Text and Command Modes--namely, to reposition the pointer and to delete and insert characters.

Immediate Mode may be used only when the WC-11 display hardware is active and the Editor is running; it is entered by typing two ALTMODES (only) in response to the Command Mode asteriskr

*$$

The Editor responds by echoing an exclamation point on the screen. The exclamation character remains *on* the screen as long as control remains in Immediate Mode.

Onca Imnediate Mode has been entered, only the commands in Table 3-3 are used. None of these commands echoes, but the text appearing on the screen is constantly refreshed and updated during the editing process. Note that no EDIT commands other than those in Table 3-3 may be used while control remains in Immediate Mode.

To return control to the display Editor's normal Command Mode at any time while in Immediate Mode, type a single ALTMODE. The Editor responds with an asterisk and the user may proceed using all normal Editing commands. (Immediate Mode commands typed at this time will be accepted as Command Mode input characters.) To return control to the monitor while in Imnediate Mode, type CTRL C.

Table 3-3
Immediate Mode Commands

| Command | Meaning |
|---|---|
| CTRLN | Advance the pointer (cursor) to the beginning of the next line (equivalent to A). |
| CTRL G | Move the pointer (cursor) to the beginning of the previous line (equivalent to -A). |
| CTRL D | Move the pointer (cursor) forward by one Character (equivalent to J). |
| CTRLV | Move the pointer (cursor) back by one character (equivalent to -J). |
| RUBOUT | Delete the Character immediately preceding the pointer (cursor) (equivalent to -D). |
| CTRLC | Return control to the monitor. |
| ALTMODE (one only)<br>(two) | Return control to Command Mode.<br>Direct control to Imnediate Mode. |
| Any other character than those above | Insert the Character as text positioned immediately before the pointer (cursor) (equivalent to I). |

## 3.8 EDIT EXAMPLE

The following example illustrates the use of some of the EDIT commands to change a program stored on tha device DK. Sections of the terminal output are coded by letter and corresponding explanations follow the example.

```
A  .R EDIT
   *ERDK:TEST1.MAC$$
   *EWDK:TEST2.MAC$$
   *R$$
B  *^/L$$
   ;TEST PROGRAM

   START :  MOV #1000,%6     ;INITIHLIZE STACK
            HOV #MSG,%0      ;POINT R0 T O MESSAGE
            JSR %7,MSGTYP    ;PRINT I T
            HALT             ;STOP
   MSG :    .ASCII/IT WORKS/
            BYTE 15
            BYTE 12
            BYTE 0
C  *B 1J 5D$$
D  *GPROGRAM$$
   *0L$$
E  ;PROGRAM*I TO TEST SUBROUTINE MSGTYF. TYFES
   ;"THE TEST FROGRAM WORKS"
   ;ON THE TEMI\IM\RMINAL$$
F  *F.ASCII/$$
   *8CTHE TEST PROGRAM WORKS$$
G  *P.BYTE-X
   *F.BYTE 0$V$$
            .BYTE 0
H  *I
            .END
   $B/L$$
   ;PROGRAM TO TEST SUBROUTINE MSGTYF. TYFES
   ;"THE TEST PROGRAM WORKS"
   ;ON THE TERMINAL

   STRRT:   MOV #1000,%6     ;INITIALIZE STHCK
            nov #MSG,%0      ;POINT RO TO MESSAGE
            JSR %7,MSGTYP    ;PRINT I T
            HALT             ;STOP
   MSG :    .ASCII/THE T E S T PROGRAM WORKS/
            BYTE 15
            .BYTE 12
            BYTE B
            E N D
I  *EX$$
```

A The EDIT program is called and prints an *. The input file is TEST1.MAC; the output file is TEST2.MAC and the first page of input is read.

B The buffer contents are listed.

C Be sure the pointer is at the beginning of the buffer. Advance pointer one character (past the ;) and delete the "TEST ".

D Position pointer after PROGRAM and verify the position by listing up to the pointer.

E Insert taxt. RUBOUT used to correct typing error.

F Search for .ASCII/ and change "IT WORKS"to "THE TEST PROGRAM WORKS".

G CTRL x typed to cancel P command. Search for ".BYTE 0" and verify location of pointer with V command.

H Insert text. Return pointer to beginning of buffer and list entire contents of buffer.

1 Close input and output files after copying the current taxt buffer as well as the rest of input file into output file. EDIT returns control to the monitor.

## 3.9 EDIT ERROR MESSAGES

The Editor prints an error message whenever one of the error conditions listed next occurs. Prior to executing any commands, the Editor first scans the entire command string for errors in command format (illegal arguments, illegal combinations of commands, etc.). ff an error of this type is found, an error message of the form:

?ERROR MSG?

is printed and no commands are executed. The user must retype the command.

If the command string is syntactically correct, execution is started. Execution errors are still possible, however (buffer Overflow, I/O errors, etc.), and if such an error occurs, a message of the form:

?*ERROR MSG*?

is printed. In this case, all commands preceding the one in error are executed, while the command in error and those following are not executed. Most errors will generally be of the syntax type and can be corrected before execution.

When an error occurs during execution of a Macro, the message format is:

?message IN MACRO?

or

?*message IN MACRO*?

depending on when it is detected.

| Message | Explanation |
|---|---|
| *CB ALMOST FULL* | The command currently being entered is within 10 characters of exceeding the space available in the Command Buffer. |
| ?CB FULL? | Command exceeds the space allowed for a command string in the Command Buffer. |
| ?*DIR FULL*? | No room in device directory for output file. |
| ?*EOF*? | Attempted a Read, Next or file searching command and no data was available. |
| ?*FILE FULL*? | Available space for an output file is full. Type a CTRL C and the CLOSE monitor command to save the data already written. |
| ?*FILE NOT FND*? | Attempted to open a nonexisting file for editing. |
| ?*HDW ERR*? | A hardware error occurred during I/O. May be caused by WRITE LOCKed device. Try again. |
| ?ILL ARG? | The argument specified is illegal for the command used. A negative argument was specified where a positive one was expected or argument exceeds the range + or - 16,383. |
| ?ILLCMD? | EDIT does not recognize the command specified; ED was not the first command issued when used to activate the display hardware. |
| ?*ILL DEV*? | Attempted to open a file on an 'illegal device, or attempted to use display hardware when none was available (it may be in use by the other job). |
| ?ILL MAC? | Delimiters were improperly used, or an attempt was made to enter an M command during execution of a Macro or an EM command while an EM was in progress. |

| Message | Explanation |
|---|---|
| ?*ILL NAME*? | File name specified in EB, EW, or ER is illegal. |
| ?*NO FILE*? | Attempted to read or write when no file is open. |
| ?*NO ROOM*? | Attempted to Insert, Save, Unsave, Read, Next, Change or Exchange when there was not enough room in the appropriate buffer. Delete unwanted buffers to create more room or write text to the output file. |
| ?*NO TEXT*? | Attempted to call in text from the Save Buffer when there was no taxt available. |
| ?*SRCIiFAIL*? | The text string specified in a Get, Find or Position command was not found in the available data. |
| ?"<>"ERR? | Iteration brackets are nested too deeply or used illegally or brackets are not matched. |

CHAPTER 4

# PERIPHERAL INTERCHANGE PROGRAM (PIP)

The Peripheral Interchange Program (PIP) is the file transfer and maintenance utility for RT-11. PIP is used to transfer files between any of the RT-11 devices (listed in Table 2-2), merge and delete files from these devices, and list, zero, and comprese device directories.

## 4.1 CALLING AND USING PIP

To call PIP from the system device type:

```
R PIP
```

in response to the dot printed by the Keyboard Monitor. The Command String Interpreter prints an asterisk at the left margin of the terminal and waits to receive a line of filenames and command switches. PIP accepts up to six input filenames and three output filenames; command switches are generally placed at the end of the command string but may follow any filename in the string. There is no limit to the number of switches which may be indicated in a command line, as long as only one Operation (insertion, deletion, etc.) is represented.

Since PIP performs file transfers for all RT-11 data formats (ASCII, Object, and image) there are no assumed extensions for either input or output files; all extensions, where present, must be explicitly specified.

Following completion of a PIP Operation, the Command String Interpreter prints an asterisk at the left margin of the teleprinter and waits for another PIP command line. Typing CTRL C at any time returns control to the Keyboard Monitor. To restart PIP, type R PIP or the REENTER command in response to the monitor's dot.

### 4.1.1 Using the "Wild Card" Construction

PIP follows the Standard file specification syntax explained in Section 2.5 (Chapter 2) with one exceptiont the asterisk character can he used in a command string to represent filenames or extensions. The asterisk (called the "wild card") in a file specification means "all". For instance, "*.MAC" means all files with the extension .MAC.

regardless of filename. "FORTN.*" means all files with the filename FORTN regardless of extension. "*.*" means all files, regardless of name or extension.

The wild card Character is legal in the following cases only (switches are explained in the next section):

1. Input file specification for the copy and multiple copy operations (i.e., no switch, /I, /B, and /A).
2. File specification for the delete Operation (/D).
3. Input and output file specifications for the renaxne Operation (/R).
4. Input and output file specifications for the multiple copy Operation (/X).
5. Input file specifications for the directory list operations (/L, /E, /F) .

Operations on files implied by the wild card asterisk are perfonned in the order in which the files appear in the directory. System files with the extension .SYS and files with bad blocks and the extension .BAD are ignored when the wild card Character is used unless the /Y switch is specified.

Examplest

| | |
|---|---|
| **,BAK/D | Causes all files with the extension .BAK (regardless of their filenames) to be deleted from the device DK. |
| **.TST=*.BAK/R | Renames all files with a .BAK extension (regardless of filenames) so that these files nw have a .TST extension (maintaining the same filenames). |
| *RK1:*.*/X/Y=*.* | Transfers all files, including system files, (regardless of filename or extension) from device DK to device RK1. |
| **,MAC,*.OBJ/L | Lists all files with .MAC and .OBJ extensions. |

## 4.2 PIP SWITCHES

The various operations which can be performad by PIP are summarized in Table 4-l. If no switch is specified, PIP assumes the Operation is a file transfer in image (/I) mode. Detailed explanations of the switches follw the table.

Table 4-1
PIP Switches

| Switch | Section | Explanation |
|---|---|---|
| /A | 4.2.2 | Copies file(s) in ASCII mode; ignores nulls and rubouts; converts to 7-bit ASCII; CTRL Z (32 octal) treatedas logical end-of-file on input. |
| /B | 4.2.2 | Copies files in formatted binary mode. |
| /C | 4.2.2 | May be used in conjunction with another switch to cause only files with current date (as designated using the monitor DATE command) to be included in the specified Operation. |
| /D | 4.2.4 | Deletes file(s) from specified device. |
| /E | 4.2.6 | Lists the device directory including unused spaces and their sizes. An empty space on a cassette or magtape directory represents a deleted file. Sequence numbers are listed for cassettes. |
| /F | 4.2.6 | Prints a short directory (filenames only) of the specified device. |
| /G | 4.2.2 | Ignores any input errors which occur during a file transfer and continues copying. |
| /I or no switch | 4.2.2 | Copies file(s) in image mode (byte by byte). This is the default switch. |
| /K | 4.2.12 | Scans tha specified device and types the absolute block numbers (in octal) of any bad blocks on the device. |
| /L | 4.2.6 | Lists the directory of the specified device, including the number of files, their dates, and the number of blocks used by each file. Sequence numbers are listed for cassettes. |
| /M:n | 4.2.1 | Used when I/O transfers involve either cassette or magtape. n reprosents the numeric position of the file to be accessed in relatfon to the physical position of the cassette or magtape on the drive. If n is positive, the tape spaces forward from its current position until either the filename or tha nth file is found; if n is negative, the tape is rewound first, and then it spacßs forward until either the filename or the nth file is found. If n is 0 (or not indicated) the tape is rewound and searched for the filename. For wild card operations, specification of /M with a positive argument will prevent the tape from rewinding between each file involved in the Operation. |
| /N:n | 4.2.7 | Used with /Z to specify the number of directory Segments (n) to allocate to the directory. |
| /O | 4.2.10 | Bootstraps the specified device (DT0, RKn, RF, DPn, DSn, DXn only). |

(continued on next page)

Table 4-1 (Cont.)
PIP Switches

| Switch | Section | Explanation |
|---|---|---|
| /Q | 4.2.2 | When used in conjunction with another PIP Operation, causes PIP to type each filename which is eligible for a wild card Operation and to ask for a confirmation of its inclusion in the Operation. Typing a "Y" causes the named file to be included in the Operation; typing anything else excludes the file. The command line is not processed until the user has confirmed each file in the Operation. |
| /R | 4.2.5 | Renames the specified file. |
| /S | 4.2.8 | Compresses the files on the specified directory device 50 thnt free blocks are combined into one area. |
| /T | 4.2.4 | Extends number of blocks allocated for a file. |
| /U | 4.2.9 | Copies the bootstrap from the apecified file into absolute blocks 0 and 2 of the specified device. |
| /V | 4.2.11 | Types the version number of the PIP program being used. |
| /W | 4.2.6 | Includes the absolute starting block and any extra directory words in the directory listing for each file on the device (numbers in octal). Used with /F, /L, or /E. |
| /X | 4.2.3 | Copies files individually (without concatenation). |
| /Y | 4.2.2 | Causes system files and .BAD files to be operated on by the command specified. Attempted modifications or deletions of .SYS or .BAD files without /Y are not done and cause the message ?NO SYS ACTION7 to be printed. |
| /Z:n | 4.2.7 | Zeroes (initializes) the directory of the specified device; n is used to allocate extra words per directory entry. When used with /N, the number of directory segments for entries may be specified. When used with cassette, /Z writes a sentinel file at the beginning of the tape; With magtape, /Z writee a volume label followed by a dummy file followed by double tape marks indicating logical end-of-tape. |

### 4.2.1 Operation8 Involving Magtape or Cassette

PIP operations involving cassette and magtape devices are handled somewhat differently than other RT-11 devices, because of the sequential nature of these devices. The last file on a cassette or magtape (the logical end-of-tape) is specially formatted ao that it marks the end of current data and indicates where new data may begin (double end-of-file for magtape, sentinel file or physical end-of-tape for cassette). Therefore, operations which designate specific block lengths (such as /T and /N) are meaningless, and unused spaces on the tape (resulting from file deletiona) cannot be filled.

PIP operations which are legal using cassette and magtape (including the bootable magtape on which the system may have been distributed) include the following: /A, /B, /D, /E, /F, /G, /I, /L, /M, /Q, /V, /W, /X, /Y, and /Z. Usually the device (CT or MT) is rewound each time an Operation is performed. Since there is no inclusive directory at the beginning of the tape the only way to access a file is to search the tape from the beginning until it is found. However, the /M:n switch is available for situations where it is not necessary or desirable to rewind the tape before each Operation. If the argument (n) is positive, the Operation indicated will not rewind the tape first, but will space forward until it finds either the nth file, the filename indicated in the command line, or the logical end-of-tape, whichever occurs first. If the argument is negative, the cassette or magtape will be rewound first and then spaced forward until the filename (or nth file, or logical end-of-tape) is found. Thus:

| | |
|---|---|
| /M:1 | means suppress rewind, begin Operation at current Position. |
| /M:-1 | means rewind tape and access the first file on it. |

Remember that when /M:n is used, n is interpreted as an octal number. /M:n must be used if it is intended that n represent a decimal number.

For example, assume the directory of a cassette on unit 1 ist

```
17-JUL-74
FILE .1      0  5-MAY-74
FILE .2      0  5-MAY-74
FILE .3      1 13-MAY-74
FILE .4      1 28-JUN-74
FILE .5      0 17-JUL-74
5 FILES, 2 BLOCKS
*
```

and the last PIP Operation involved FILE.4, leaving the cassette positioned at the and of FILE.4. To access FILE.2, the next Operation (for example, deleting FILE.2) could use the /M construction:

```
*CT1:DUM/M:-2/D
```

In this case, the cassette rewinds first, then epaces forward from its currunt position to the second file in sequence and deletes it. (In a delete Operation, the dummy filename is necessary; otherwise, a non-file structured delete is performed and the tape is zeroed. See Section 4.2.4).

Another useful application of the /M switch involves a case where a nuraber of files are to be created on a magtape or caasette. Using the construction:

```
*MT:*.*/X=FILE.1,FILE.2.../M:1000
```

prevents a rewind from occurring before each new file is created on the tape. Normal Operation (when creating a new file on magtape or cnssette) is to rewind, then search the tape for the logical end. If a file with the eame name as the one being created is encountered, it is deleted and the new file is opened at the logical end of the tape. The /M:1000 command first causes the tape to space forward until it reaches the logical end-of-tape, (assuming less than 1000 (octal) files on the tape), at which point the next file is entered, and so on. If the tape were already positioned at the end of the tape, an

/M:1 would suffice to cause the new file to be written there. Note that creation of a new file with the /M switch can result in several files with the same name on the same taps; those files occurring before the tape position are not searched for duplication prior to the creation of the new file.

RT-11 magtapes sometimes contain a dummy file at the beginning of the tape, which is written when the tape ie initialized with the /Z switch. This file shows up in extended directories (/E) as an <UNUSED> entry in the first file position. Deleted files on magtape or cassette do not show up in /F or /L directory listings, but must always be considered when the /M:n switch is used. Care must always be taken to use a /E directory when counting file position prior to using that position as an /M:n argument; <UNUSED> files must be counted as files on the tape.

For examplet

```
R PIP
*MT0:/E                                  Extended directoryt shows
11-SEP-74                                absolute file positions.
< UNUSED >     0
A   .  MAC    40  11-SEP-74
B     ,MAC.   15  11-SEP-74
<  UNUSED  >   2
D      .MAC     2  11-SEP-74
3  FILES, 5 7  BLOCKS

*MT0:/L                                  Normal directory; does
11-SEP-74                                not accurately display
A     ..MAC    4 0 11-SEP-74             file positions.
B       MAC    15  11-SEP-74
D     . MAC     2  11-SEP-74
3 FILES, 5 7  BLOCKS
```

If the user wished to access file A.MAC on the magtape in the example above , /Mt-2 must be used (/M:-1 would access the first empty file). Likewise, B.MAC is accessed with /M:-3. Rewind can also be suppressed for cassette and magtape as input devices by specifying a very large number in conjunction with wild card transfers from magtape or cassette.

```
**.*=MT0:*.*/M :2000/X
```

This transfers all files from MT0: to DK: without rewinding between each file. The argument 2000 is an arbitrarily large number; *any* number larger than the actual number of files on the tape will suffice.

The most common method for spacing to the end of the tape is:

```
*DUMMY=MT0:DUMMY/M:2000
?FIL NOT FND?
```

where DUMMY is a file name which does not exist on the tape. Note that an error message is printed when the end of the tape is reached.

Directory listings of magtapes include the length of each file in 256 (decimal) word blocke. In cassette directories, however, sequence numbers rather than block numbers are printed. Sequence numbers indicate the sequential ordering of a file in cases where it has been continued on more than one cassette. In the example cassette directory listing (at the beginning of this section), the numbers in the middle column represent sequence numbers; both FILE.3 and FILE.4 are the second Segments of continued files. All files on cassette are initially assigned a sequence number of 0 (meaning this is the first Segment of the cassette file, not that the file has no length). The sequence number is automatically updated whenever the file must be continued as a result of a full cassette.

During I/O transfer Operation8 involving cassette, if the cassette ie full before the transfer has finished, the messaget

CTn: PUSH REWIND OR MOUNT NEW VOLUME

is printed; n represents the number of the drive (0 or 1) on which the current cassette is mounted. If the cassette rewind button is subsequently pushed, an error meesage is typed (IN or OUT ERR) and the tape is rewound.

To continue an output Operation, mount a new cassette (which has been properly formatted as described in Section 4.2.7) on the same drive. The new cassette is rewound automatically and a file is opened *on* it under the same name and extension; the sequence *number* in its directory is updated to reflect the continuation, and the transfer continues.

If the message occurs during an input Operation, mount the cassette containing the continued portion of the file on the drive; the cassette is rewound first. PIP then looks for a file with the same name and extension and the proper sequence number and continues the input Operation. The message is repeated if the next segment is not found.

For example:

```
*CT0:FILE.AGA=DT1:ASC.MAC,DK:BALOR.MAC/A
CT0: PUSH REWIND OR MOUNT NEU VOLUME
```

This copies in ASCII mode the file ASC.MAC from DECtape 1 and BALOR.MAC from device DK and combines them under the name FILE.AGA on CTO. The cassette runs out of room and requeats that a new one be mounted. The Operation continues automatically when the second cassette has been mounted.

A directory of the second cassette in the above Operation is next requested; note that the sequence number of FILE.AGA is 1, signifying it ie the aecond part of a continued file.

```
*CT0:/L
23-MAY-74
TRA    .BIN     0 16-FEB-74
FILE   .AGA     1 23-MAY-74
2 FILES, 1 BLOCKS
*
```

(The number of blocks in a cassette directory simply represents the total of sequence numbers in the directory.)

Any cassette mounted in response to a continuation message MUST have been previously initialized at some time as described in Section 4.2.7.

If a full cassette is mounted or an attempt is made to access some file on it that does not exist, the continuation message recurs. The Operation may be continued by mounting another cassette.

Note that if an attempt is made to access a file which has a non-zero sequence number (during some Operation which is not a continuation of an Operation), the file will not be found.

To copy multiple files to a cassette using a wild card command, use the following:

```
*CTn:*.*=DEV:*.*/X/M:1      (rewind is inhibited)
```

Continue to mount new cassettes in response to the PUSH REWIND OR MOUNT NEW VOLUME message. Do not ahort 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 the following:

```
*DEV:*.*=CTn:*.*/X/M:1000      (rewind is inhibited)
```

Whenever a continued volume is detected, the PUSH REWIND OR MOUNT NEW VOLUME message will appear, until the entire file has been copied (assuming that each sequential cassette is mounted in response to each occurrence of the message). Whenever PIP has copied the final section of a continued file, it will return to command level. To copy the remaining files on that cassette, reissue the command:

```
*DEV:*.*=CTn:*.*/X/M:lOOO
```

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.

If the end of a tape is reached during a magtape I/O Operation, an IN or OUT ERR message is printed. In the case of an output Operation, the magtape backspaces and deletes the partial file by writing logical end of tape over the file's header label. The Operation must then be repeated using another magtape.

If CTRL C ie typed during any output Oparation to cassette or magtape, an end-of-tape or sentinel file is not written on the tape first. Consequently, no future enters may occur to the tape unless one of two recovery procedures is followedr

1. Transfer all good files from the bad tape to another tape and zero the bad tape in the following manner:

   ```
   *devl:*.*/X=dev0:filel,file2,...filen/M:1000
   *dev0:/Z
   dev0:/Z ARE YOU SURE ?
   ```

   This causes a logical end-of-tape to be written onto the bad tape and makes it again available for use.

2. Determine the sequential number of the file which was interrupted and use the /M construction to enter a replacement file (either a new file or a dummy file). Assuming the bad file is the 4th file on the tape, use a command line of this construction:

```
*dev0:file.new=file.dum/M:-4
```

A logical end-of-tape now exists on the tape, making it available for use.

Since magtapes and cassettes are not random access devices, each unit can have only one file accessed at a time. Avoid PIP command strings which specify the same unit number for both input and output, since a lose of information can occur. For examplet

```
*CT0:FILE1.MAC=CT0:FILE1.MAC
?FIL NOT FND?
*
```

The result of this Operation is to delete FILE1.MAC before the error message is printed, and the tape label structure may be destroyed.

Recovery procedures for errors caused by bad tapes are described in RT-11 Software Support Manual.

This page intentionally blank.

### 4.2.2 Copy Operation8

A command line without a switch causes files to be copied onto the destination device in image mode (byte by byte). This Operation is used to transfer memory image (save format) files and any files other Man ABC11 or formatted binar-y. For example:

```
*ABC<XYZ
```
Makes a copy of the file named XYZ on device DK and assigns the name ABC. (Both files exist on device DK following the operation).

```
*SY:BACK.BIN=PR:/I
```
Copies a tape from the Papertape reader to the system device in image mode and assigns it the name BACK.BIN.

The /A switch is used to copy file(s) in ASCII mode as followsr

```
*DT1:F1<F2/A
```
Copies F2 from device DK onto device DT.1 in ASCII mode and assigns the name Fl.

Nulls and rubouts are ignored in an ASCII mode file transfer. CTRL Z (32 octal) is treated as logical end-of-file if eacountered in the input file.

The /B switch is used to transfer formatted binary files. The formatted binary copy switch should be used for .OBJ files produced by the asaembler or FORTRAR and for .LDA files produced by the Linker. For example:

```
*DK:FILE.OBJ<PR:/B
```
Transfers a formatted binary file from the Papertape 'reader to device DK and assigns the name FILE.OBJ.

When performing formatted binary transfers, PIP verifies checksums and prints the message ?CHK SUM? if a checksum errbr occurs.

If neither /A nor /B is used in a copy Operation that involves a paper tape device, the size of the output file in the Operation depends upon the memory size of the System. The transfer mode defaults to image mode and PIP attempts to do a single read to fill its input buffer. When a read from the paper tape reader encounters end-of-tape, no count of words transferred can be returned; PIP assumes its input buffer is full and copies it to the output device. The output file size thus depends upon the input buffer size, which is determined by the memory size of the System. The output file will have several blocks of zeroes after the end of the paper tape image. If copying to the punch, large amounts of blank tape will be punched after the input tape image is output. The extra length is harmless, but can be avoided by use of /A or /B. Image mode files (for example, .SAV files) cannot reliably be transferred to or from paper tape.

To combine more than one file into a single file, use the following format:

```
*DK:AA<DT1:BB,CC,DD/I
```
Transfers files BB, CC and DD to device DK as one file and assigns this file the name AA.

*DT3:MERGE-QT2:FILE2,FILE3/A
Merges ASCII files FILE2 and FILE3 on DT2 into one ASCII file named MERGE on device DT3.

Errors which occur during the copy Operation (such as a parity error) cause PIP to output an error message and return for another command string.

The /G switch is used to copy files but ignore all input errors. For example:

*ABC<DT1:TOP/G
Copies file TOP in image mode from device DT1 to device DK and assigns the name ABC. Any errors during the copy Operation are ignored.

This page intentionally blank.

```
*DT2:COMB<DT1:F1,F2/A/G
```

Copies files F1 and F2 in ASCII mode from device DT1 to device DT2 as one file with the name COMB. Ignores input errors.

The wild card construction may be used for input file specifications during copy operations. Be sure to use the /Y switch if system files (.SYS) are to be copied. For example:

```
*DT1:PROG1<*.MAC
```

Copies, in image mode, all files with a .MAC extension from device DK to device DT1 and combines them under the name PROG1.

```
**.*=DT3:*.*/G/Y/X
```

Copies to device DK, in image mode, all files (including .SYS files) from device DT3; ignores any input errors.

If only files with the current date are to be copied (using the wild card construction), the /C switch must also be used in the command line. For example:

```
*DT2:NN3=ITEM1.*/C,ITEM2/A
```

Copies, in ABC11 mode, all files having the filename ITEM1 and the current date, (the date entered using the monitor DATE command) and copies ITEM2 (regardless of its date) from device DK to device DT2 and combines them under the name NN3.

```
*DT3:*.*=*.*/C/X
```

Copies all files with the current date from DK to DT3. Note that commands of this nature are an efficient way to backup all new files after a session at the computer.

The /Q switch is used in conjunction with another PIP Operation and the wild card construction to list all files and allow the user the opportunity to confirm individually which of these files should be processed during the wild card expansion. Typing a "Y" causes the named file to be processed; typing anything else excludes the file. For example:

```
**.OBJ<DT1:*.OBJ/Q/X
FIRST.OBJ?Y
GETR.OBJ?
BORD.OBJ?
CARJ.OBJ?Y
```

Copies the files FIRST.OBJ and CARJ.OBJ to the disk in image mode from DECtape 1 and ignores the others.

The file allocation scheme for RT-11 normally allows half the entire largest available space or the second largest space, or a maximum sise (a constant which may be patched in the RT-11 monitor; see the RT-11 System Generation Manual), whichever is largest, for a new file. The user can, using the [n] construction explained in Chapter 2, force RT-11 to allow the entire largest possible space by setting n=177777. If n is set equal to any other value (other than 0 which is default and gives the normal allocation described first above), that size will be allocated for the file.

Therefore, assume that the directory for a given device shows a free area of 200 blocks and that PIP returns an ?OUT ER? message when a transfer is attempted to that device with a file which is longer than 100 blocks but lese than 200 blocks. Transfers in this situation can be accomplished in either of two waysr

1. Use the [n] construction on the output file to specify the desired length (refer to Chapter 2, Section 2.5 for an explanation of the [n] construction).

2. Use the /X switch during the transfer to force PIP to allocate the correct number of blocks for the output file. This procedure will operate correctly if the input device is DECtape or disk.

For example, assume that file A is 150 blocks long and that a directory listing shows that there is a 200 block <unused> space on DT1:

```
.R FIP
*DT1:A=A
?OUT ER?
```

File longer than 100 blocks.

```
*DT1:A[150]=A
```

or

```
*DT1:A=A/X
```

Either command causes a correct transfer.

### 4.2.3 Multiple Copy Operation8

The /X switch allows the transfer of several files at a time onto the destination device as individual files. The /A, /G, /C, /Q, /B and /Y switches can be used with /X. ff /X is not indicated, all output files but the first will be ignored.

Examplesr

```
*FILE1,FILE2,FILE3<DT1:FILEA,FILEB,FILEC/X
```

Copies, in image mode, FILEA, FILEB and FILEC from device DT1 to device DK as separate files called FILE1, FILE2 and FILE3, respectively.

```
*DT2:F1.*=F2.*/X
?NO SYS ACTION?
*
```

Copies, in image mode, all files named F2 (except files with .SYS or .BAD extensions) from device DK to device DT2. Each file is assigned the filename Fl but retains its original extension.

```
*DT1:*.*=DT2:*.*/X
?NO SYS ACTION?
```

Copies, in image mode, all files on device DT2 to device DT1 (except files with .SYS or .BAD extensions); the files are copied separately and retain the same *names* and extensions.

```
*DT1:FILE1,FILE2<FILEA.*/A/G/X
```

This command line assumes there are two files with the filename FILEA (and any extension excluding .SYS or .BAD extensions) and copies these files in

ASCII mode to device DT1. The files are tranaferred in the order they are found in the directory; the first file found is copied and assigned the name FILE1, and the second is assigned FILE2. ff there is a third, it is ignored and a fourth causes an ?OUT FIL? error.

```
*DTB : *. SYS=*. SYS/X/Y
```

Copies all system files from devfce DK to device DTO.

File transfers performed via normal Operation8 place the new file in the largest available *area* on the disk. The /X switch, however, places the copied files in the first free place large enough to accommdate it. Therefore, the /X ewitch should be used whenever possible (i.e., when no concatenation is desired) as an aid to reducing disk fragmentation.

```
*A=B
```

and

```
*A=B/X
```

perform the same operation; however, using the second construction whenever possible increases the system disk-usage efficiency.

For example, assume the directory of DT1 ie:

```
 9-MAY-74
MONITR. SYS 32  5-MAY-74
< UNUSED >    2
P  R  . SYS    2  5-MAY-74
< UNUSED >  528
2 FILES, 34 BLOCKS
 530 FREE BLOCKS
```

Ta copy the filePP.SYS (2 blocks long) from DK to DT1, the command:

```
*DT1 : PP. SYS=PP. SYS/Y
```

can be entered, and the new directory is:

```
 9-MAY-74
MONI TR. SYS    3  2   5-MAY-74
< UNUSED >     2
PR   .  SYS     2  5-MAY-74
PP    . SYS     2  9-MAY-74
< UNUSED > 5  2  6
3 FILES, 36 BLOCKS
 528 FREE BLOCKS
```

If the command:

```
*DT1: PP. SYS=PP. SYS/Y/X
```

had been entered, the new directory would appear:

220,

```
 9-MAY-74
MONITR.SYS    3 2  5-MAY-74
PP      SYS     2  9-MAY-74
PR     .SYS     2  5-MAY-74
<UNUSED>5  2  8
3 FILES, 36 BLOCKS
 528 FREE BLOCKS
```

### 4.2.4 The Extend and Delete Operation8

The **/T** switch is used to increase the number of blocke allocated for the specified file. The file associated with the **/T** switch must be followed by a **numeric** argument of the form [n] where n **is** a **decimal** number indicating the number of blocke to be allocated to **the file** at the completion of the extend Operation.

The format of the **/T** ewitch ist

**dev:filnam.ext[n]=/T**

A file **can** be extended in this **manner** only if it **is** followed by an unused area of **sufficient size** (on whichever device it ie located) to **accommodate** the additional length of the extended file. It may be necessary to **create** this space by moving other **files** on **the** device using the **/X** switch.

Specifying the **/T** switch in conjunction with a file that does not currently exist creates a file of the designated length.

*Error* messages are printed if the **/T** c**ommand** makes the specified file **smaller (?EXT** NEG?) or if there **is insufficient** space **following** the file **(?ROOM?).**

Examples:

| | |
|---|---|
| *ABC[200]=/T | Assigns 200 **blocks** to file ABC on device DK. |
| *DT1:XYZ[100]K/T | Assigns 100 **blocks** to the file named **XYZ** on device **DTl.** |

The **/D** switch **is** used to delete one or more files from the epecified device. The wild **card** Character **(*) can** be used in conjunction with **this** command.

Only six files **can** be specified in a delete Operation if **each** file to be deleted is individually named (i.e., if the wild **card character** ie not used).

A **cassette** or **magtape** may be initialized by indicating the **/D** switch and omitting any filenames. *For* examplet

```
*MT:/D
*CT:/D
```

Both devices are zeroed. This **is** not the **case** with the other RT-11 devices, where **omission** of a **filename causes** no **action** to occur.

When a file is deleted on block-replaceable devices, the information is not destroyed. The file name is merely removed from the directory. If a file has been deleted but not overwritten, it can be recovered with the /T switch by specifying a command of the formt

```
filena.ext[n]=/T
```

where filena.ext is the *name* desired and n is the length of the deleted file. For examplet

```
*DT1:/E
 4-JUN-74
A      .MAC      18   3-JUN-74
B      .MAC      17   3-JUN-74
C      .MAC      19   3-JUN-74
< UNUSED >  510
3 FILES, 54 BLOCKS
 510 FREE BLOCKS


*DT1:B.MAC/D

*DT1:/E
 4-JUN-74
A      .MAC    18  3-JUN-74
< UNUSED > 17
C      .MAC    19  3-JUN-74
< UNUSED > 510
2 FILES, 37 BLOCKS
 527 FREE BLOCKS
```

File B.MAC could now be recovered by:

```
*DT1:B.MAC[17]=/T
```

The /T switch looks for the first unused area large enough to accommodate the requested file length. If the file to be recovered is in the first area large enough to accommodate the size specified, the preceding command is sufficient. If not, all larger unused spaces preceding the desired file must be given dummy names before the recovery can be made.

For instance, assume the previous example with the exception that A.MAC has a 33 block unused file before it, so that the directory looks liket

```
*DT1:/E
 4-JUN-74
<  UNUSED   >   33
A       MAC     18  3-JUN-74
<  UNUSED  >  17
C     ..MAC     19  3-JUN-74
<UNUSED>4  7  7
2 FILES, 37 BLOCKS
 527 FREE BLOCKS
```

A recovery of B.MAC would requirer

```
*DT1 : DUMMYC 33 ]=/T
*DT1:B.MAC[17]=/T
```

If the 33 block unused area was not named prior to B.MAC, the first 17 blocke of the 33 block area would become B.MAC. Note that magtape and cassette files cannot be recovered once deleted.

Examplest

| | |
|---|---|
| *FILE1.SAV/D | Deletes FILE1.SAV from device DK. |
| *DT1:*.*/D | Deletes all files from device DT1 except those with a .SYS or .BAD extension. If there is a file with a .SYS or .BAD extension, the message ?NO SYS ACTION? is printed to remind the user that these files have not been deleted. |
| **.MAC/D | Deletes all files with a .MAC extension from device DK. |
| *DT1:B1, DT2:R1,DT3:AA/D | Deletes the files specified from the associated devices. |
| *RK1:*.*/D/Y | Deletes all files from device RK1. |

### 4.2.5 The Rename Operation

The /R switch is used (in a manner similar to the multiple copy command described in Section 4.2.3) to rename a file given as input with the associated name given in the output specification. There must be an equal number of input and output files and they must reside on the same device, or an error message will be printed. The /Y switch must be used in conjunction with /R if .SYS files are to be renamed.

The Rename command is particularly useful when a file on disk or DECtape contains bad blocks. By renaming the file with a .BAD extension, the file pennanently resides in that area of the device so that no other attempts to use the bad area will occur. Once a file is given a .BAD extension it cannot be moved during a compress Operation. .BAD files are not renamed in wild card operations unless /Y is used.

Examplest

| | |
|---|---|
| *DT1:F1, X1<DT1:F0, X0/R | Renames FO to Fl and XO to X1 on device DT1. |
| *FILE1.*<FILE2.*/R | Renames all files on device DK with the name FILE2 (except files with .SYS or .BAD extension) to FILE1, retaining the original extensions. |

/R cannot be used with magtape or cassette.

### 4.2.6 Directory List Operation8

The /L switch lists the directory of the specified device. The listing contains the current date, all files with their associated creation dates, total free block8 on the device if disk or DECtape, the number of files listed, and number of block8 used by the files

(sequence number for cassette). File lengths, number **of blocks** and number of files are indicated as **decimal** values. If no output device is specified, the directory is output to the terminal **(TT:).**

Examples:

```
*DT1:/L
 1-AUG-74
MONITR. SYS    3  2   5-MAY-74
PP   .  SYS     2   9-MAY-74
PR   .  SYS     2   5-MAY-74
F2      REL    15
MERGE .         2
COMB            2
6 FILES, 55 BLOCKS
 509 FREE BLOCKS
```

Outputs complete directory of device DT1 to the terminal.

```
*DIRECT=DT3:/L
```

Outputs complete directory of device DT3 to a file, **DIRECT,** on the device DK.

```
**. MAC/L
 1-AUG-74
VTMAC MRC      7  22-JUL-74
FILE2 . MAC    1
2 FILES, 8 BLOCKS
3728 FREE BLOCKS
*
```

**Lists o** the terminal a **directory** of files on device DX with the extension **.MAC.**

```
*CT1:/L
10-SEP-74
PAT1 F O R     0  10-SEP-74
PAT2 . F O R   0  10-SEP-74
IHUL . OBJ     0  10-SEP-74
SQRT    FTN    0  10-SEP-74
4 FILE;, 0 BLOCKS
```

Lists all files on cassette drive 1. For cassette only, the third column represents the sequence number. In this example, the first **segment** of **each** file is on this cassette. (See **Section** 4.2.1.)

The **/E switch** lists the entire directory including the unused areas and their **sizes** in **blocks (decimal);** an empty **space** appears in cassette and **magtape** directories to designate a deleted file.

Examplesr

```
*/E
 9-SEP-74
BATCH  .HLP   2  23-AUG-74
CHESS    .SAV 20 23-AUG-74
PAT1   .FOR   10 23-AUG-74
IRAD50.MAC     8 23-AUG-74
    .
```

Outputs to the terminal a complete directory of the device DK including the **size** of unused areas.

```
< UNUSED >  2
TRIG  .OBJ    2  6-SEP-74

STP   .OBJ    2  6-SEP-74
BAC   .OBJ    2  6-SEP-74
< UNUSED >   20


LIBR1 .OBJ  137  6-SEP-74
DIRECT        1  9-SEP-74
< UNUSED >  230
254 FILES, 4280 BLOCKS
 498 FREE BLOCKS
```

```
*LP:=CT1:/E
11-SEP-74
A      .MAC     0 11-SEP-74
A      .MAC     0 11-SEP-74
B      .MAC     0 11-SEP-74
3 FILES, 0 BLOCKS
```

Outputs to the line **printer** a **complete** directory of cassette **drive** 1. **0's** represent **segment** numbers.

The **/F** switch liste **only filenames,** omitting the file lengths and associated dates.

Examplesr

```
*DT0:/F
TRACE . MAC
CARGO . REL
BMAP    OBJ
AAR
```

Outputs a filename directory of the device DTO to the terminal.

```
*LP:=CT1:/F


A      ,MAC
A      .MAC
B      .MAC
```

Outputs a filename directory of the device **CT1** to the line **printer.**

The **/L, /E** and **/F** commands have no **effect** on the files of the **specified** device. If the **/W** switch is used in conjunction with the **/L** or **/E** switches, the absolute starting block of the file and extra words (in octal) will **be** included in the listing (for all but cassette and magtape). For example:

```
*RK1:/L/W
10-SEP-74
DSQRT .OBJ     1 10-SEP-74     16       0
MAIN  .OBJ     1 10-SEP-74     17       0
BASICR.OBJ    11 10-SEP-74     20       0
OTSV2 .OBJ     3 10-SEP-74     33       0
```

The first three columns indicate the filename and extension, block length, and date. The fourth column **shows** the absolute starting block (in octal), and the fifth column **shows** the contents of **each** extra word per directory entry (in octal). **(This** is allocated **using** the **/Z:n** switch; see **Section** 4.2.7.)

Uaing the /L, /E, or /F switch in conjunction with a device and filename causes the filename, and optionally the date and file length, to be output rather than a directory of the entire device. For example:

```
*F1.SAV/L
```

causes:

```
 4-JUN-74
F1    .SAV    18    4-JUN-74
3718 FREE BLOCKS
*
```

to be output, providing tho file exists on device DK.

Directories are made up of Segments which are two blocks long. Full directory listings with multiple segments contain blank lines as Segment boundaries.

### 4.2.7 The Directory Initialization Operation

The /Z switch clears and initializes the directory of an RT-11 directory-structured device and writes logical end-of-file to a cassette or magtape device. The /Z Operation must always be the first operation performed on a new (that is, previously unused) device. The form of the switch is:

```
/Z:n
```

where n is an optional octal number to increase the size of each directory entry on a directory-structured device. If n is not specified, each entry is 7 words long (for filename and file length information) and 70 entries can be made in a directory Segment. 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:

```
507/((# of extra words)+7)
```

For example, if the switch /Z:1 is used, 63 entries can be made per Segment.

More information concerning the format of directory entries is supplied in Chapter 3 of the RT-11 Software Support Manual.

When /Z is used, PIP responds as followsr

```
device/Z ARE YOU SURE ?
```

For example:

```
*DT1:/Z
DT1:/Z   RRE YOU SURE ?
```

Answer Y and a carriage return to perform the initialization. An answer beginning with a Character other than Y is considered to be no.

Example:

```
*DT1:/Z
DT1:/Z   ARE YOU SURE ?Y<CR>
*
```

Zeroes the directory on device DT1 and allocates no extra words for the directory.

The /N switch is used with /Z to specify the number of directory segments for entries in the directory. The form of the switch is:

/N:n

where n is an octal number less than or equal to 37. Initially RT-11 allocates four directory Segments, each two blocks (512 words) long. Refer to Chapter 3 of the RT-11 Software Support Manual for more information.

Example:

| | |
|---|---|
| *RK1:/Z:2/N:6 | Zeroes the directory on device RK1, allocates two extra words per directory entry and allocates six directory segments. |

### 4.2.8 The Compress Operation

The /S switch is used to compress the directory and files on the specified device, condensing all the free (unused) blocks into one area. Input errors are reported on the console terminal unless the /G switch is used; output errors are always reported. In either case, the compress continues. /S can also be used to copy DECtapes and disks. When DT, DP, or RK devices are copied, /S serves to both initialize the volume and to copy directory and files. When DX disks are copied, however, the output diskette must first be initialized using /Z to write the appropriate volume identification. (It is important to note that the /S switch destroys any previous directory on the output device. The new directory on the output device has the same number of Segments as the directory on the input device.) /S does not copy the bootstrap onto the volume.

To increase the nunher of directory blocks in a two-volume compress (that ie, from one volume to another rather than from one volume to itself), use the /N:n switch in conjunction with the /S switch (any attempts to decrease the directory size are ignored).

/S does not move files with the .BAD extension. This feature provides protection against reusing bad blocks which may occur on a disk. Files containing bad blocks can be renamed with the .BAD extension and are then left in place when a /S is executed.

If a compress Operation is perfonned on the system device, the message :

?REBOOT?

is printed to indicate that it may be necessary to reboot the System. If .SYS files were not moved during the canpress operation, it is not necessary to reboot the System.

NOTE

Rebooting the system in response to the ?REBOOT? warning message should ONLY be done AFTER the Operation which generated the message is complete. ?REBOOT? does not sfgnify that the system should be

rebooted immediately; the user should wait for the "*" signifying that PIP is ready for another command before rebooting.

If the command attempts to compress a large device to a smaller one, an error results and the directory of the smaller device is zeroed. If a device is being compressed in place, input and output errors are reported on the terminal and the Operation continues to completion.

**Examples:**

```
*SY:/S
?REBOOT?
```
Compreases the files on the system device SY:

```
*DT1:A<DT2:/S
```
Transfers and compresses the files from device DT2 to device DTl. Device DT2 is not changed. The filename A is a dummy specification required by the Command String Interpreter.

/S cannot be used when a foreground job is presentt a ?FG PRESENT? error message results if this is attempted.

### 4.2.9 The Bootstrap Copy Operation

The bootstrap copy switch (/U) copies the bootstrap portion of the specified file into absolute blocks 0 and 2 of the specified device.

**Examples:**

```
*DK:A<DK:MONITR.SYS/U
```
Writes the bootstrap file MONITR.SYS in blocks 0 and 2 of the device DK. A is a dummy f ilename.

```
*DT:MONITR.SYS/X/Y=RK:DTMNSJ.SYS
*DT:A=RK:DTMNSJ.SYS/U
```
Writes the Single-Job DECtape Monitor to device DT0 and then writes the bootstrap into blocks 0 and 2 (the bootstrap is written from disk rather than DECtape because disk is faster).

### 4.2.10 The Boot Operation

The boot switch reboots the System, reinitializing monitor tables and returning the system to the monitor level. The boot switch performs the same Operation as a hardware bootstrap.

Example:

```
*DK:/O
```
Reboots the device DK.

ff a boot switch is specified on an illegal device, the message:

```
?BAD BOOT?
```

is printed. Legal devices are DTO, RK0-RK7, RP, SY, DK, DPO-DP7, DX0-DX1, and DSO-DS7. Note that /o is illegal if a foreground job is present; the ?FG PRESENT? error message results. The user must abort the foreground job and unload it before using /o.

### 4.2.11 The Version Switch

The Version switch (/V) Outputs a version number message (representing the Version of PIP in use) to the terminal using the form:

```
PIP V02-XX
```

The rest of the command line, if any, is ignored.

### 4.2.12 Bad Block Scan (/K)

The bad block switch (/K) scans the specified device and types the absolute block numbers of those blocks on the device which return hardware errors. The block numbers typed are octal; the first block on a device is 0(8). Note that if no errors occur, nothing will be output. A complete scan *of* a disk pack takes several minutes.

**Example:**

```
*RK2:/K                    Scan disk drive 2 for bad blocke.
BLOCK    140 IS BAD

*RK:/K                     Scan drive 0. No blocks are bad.
*
```

#### 4.2.12.1 Recovery from Bad Blocks

As a disk ages, the recording surface wears. Eventually unrecoverable I/O errors occur during attempts to read or write a bad disk block. PIP protects against usage of bad disk areas by ignoring files with a .BAD extension (unless the /Y switch is used). Once a bad block is uncovered in an I/O Operation, it can be located using the /K switch and a .BAD file can be created which encompasses the bad block.

When a hardware I/O error is detected, the recovery procedure is as follows:

1. Use the PIP /K switch to scan the device and print on the terminal the absolute block numbers (in octal) of the bad blocks. For example:

```
R PIP
*RK1:/K
BLOCK    7723 IS BAD
*
```

2. Obtain an extended directory with the /W switch, showing the starting block numbers of all the files on the disk.

3. ff a bad block occurs in a file with valuable information, copy the file to another file using the /G switch. In most cases, only 1 bit (Character) of the file is affected.

4. If the file is small, it can then be renamed with a .BAD extension to prevent further use of that disk area.

5. If the file is large or the bad block occurs in an empty area, a l-block .BAD file can be created using the /T switch as follows:

   a. Delete the bad file (if any).

   b. If the bad block is at block n of the freo area, create a file of length n-l with the /T switch. Remember that there must be no spaces larger than n-l blocks before the desired one (refer to Section 4.2.4). Also note that the block numbers printed in the /K and /W operations are octal, while the argument to the /T Operation is decimal.

   c. Create a l-block .BAD file with the /T switch to cover the bad block.

   d. Delete any temporary files created during the Operation.

   For example, assume the extended directory is:

```
.
.
NEWSRC.BAT      8 11-SEP-74    6203
RTTEMP.BAT     27 11-SEP-74    6213
PIP    .MAC   150 12-SEP-74    6246
. UNUSED .    154
VERIFY.SAV      3              6726
. UNUSED .    300
PIP    .OBJ    15 12-SEP-74    7405
MKPIP  .CTL     1 12-SEP-74    7424
MKV2RK.CTL      U 12-SEP-74    7425
VTLIB  .OBJ    10 12-SEP-74    7431
. UNUSED .    150
A               4 12-SEP-74    7671
PIP    .LST   300  3-SEP-74    7675 Block 7723 (octal) of
.                                   PIP.LST is bad.
.
```

and a bad block is detected at block 7723 (octal) of the file PIP.LST. To recover, make a copy, ignoring the error, and delete the bad file:

```
*RK1:PIPA. LST=RK1:PIP. LST/G
*RK1:PIP. LST/D
```

The directory now readst

```
.
.
NEWSRC.BAT      8 11-SEP-74    6203
RTTEMP.BAT     27 11-SEP-74    6213
PIP    .MAC   150 12-SEP-74    6246
```

```
•  UNUBCD  >  154
VERIFY.SAV     3                          6726
PIPA  .LST   300  18-SEP-74    6751
PIP   . OB3   15  12-SEP-74    7405
MKPIP .CTL     1  12-SEP-74    7424
MKV2RK.CTL     4  12-SEP-74    7425
VTLIB .OBJ    10  12-SEP-74    7431
<  UNUSED  •   150
4              4  12-SEP-74    7671
   .
   .
```

An unused area following A contains block 7723 **(octal),** which is bad. Continuing In **PIP:**

```
*RK1:TEMP. 002[ 154 ]=/T
*RK1: TEMP. 003[ 150 ]=/T
*RK1 : TEMP. 004[ 22 ]=/T
```

This fills the unused areas with temporary files. Specifying **TEMP.004** with a length of 22 **blocks** makes the file just long enough to precede the bad block (i.e., 7675 (octal) and 22 **(decimal)** equal 7723, which would be the starting block **number** of the next file created). The directory now contains:

```
   .
   .
NEWSRC.BAT     B  11-SEP-74    6203
RTTEMP.BAT    27  11-SEP-74    6213
PIP   .MAC   150  12-SEP-74    6246
TEMP  ,002   154  18-SEP-74    6474
VERIFY.SAV     3                          6726
PIPA  .LST   300  18-SEP-74    6731
PIP   ,OBJ    15  12-SEP-74    7405
MKPIP .CTL     1  12-SEP-74    7424
MKV2RK.CTL     4  12-SEP-74    7425
VTLIB .DBJ    10  12-SEP-74    7431
TEMP , 0 0 3 156  18-SEP-74    7443
A              4  12-SEP-74    7671
TEMP  .004    22  18-SEP-74    7675
   .
   .
```

Continuing with **PIP:**

```
*RK1:F 1 LE.BAD[1]=/Y/T                    Create a bad file.
```

The directory **now** containst

```
   .
NEWSRC.BAT     8  11-SEP-74    6203
RTTEMP.BAT    27  11-SEP-74    6213
PIP   .MAC   150  12-SEP-74    6246
TEMP  ,002   154  18-SEP-74    6474
VERIFY.SAV     3                          6726
PIPA  .LST   300  18-SEP-74    6731
PIP   .OBJ    15  12-SEP-74    7405
MKPIP .CTL     1  12-SEP-74    7424
MKV2RK.CTL     4  12-SEP-74    7425
VTLIB .OBJ    10  12-SEP-74    7431
TEMP  .003   150  18-SEP-74    7443
A              4  12-SEP-74    7671
```

```
TEMP  .004    22 18-SEP-74   7675
FILE  .BAD     1 18-SEP-74   7723
```

Bad block is here.

Next delete all temporary files and rename PIPA.LST to PIP.LST. The final directory now contains:

```
NEWSRC.RAT     8 11-SEP-74   6203
RTTEMP.BAT   2 7 11-SEP-74   6213
PIP   .MAC 1 5 0 12-SEP-74   6246
< UNUSED > 1 5 4
VERIFY.SAV     3             6726
PIP   .LST 3 0 0 18-SEP-74   6731
PIP   .OBJ   1 5 12-SEP-74   7405
MKPIP .CTL     1 12-SEP-74   7424
MKV2RK.CTL     4 12-SEP-74   7425
VTLIB .OBJ    10 12-SEP-74   7431
< UNUSED > 150
A              4 12-SEP-74   7671
<  UNUSED  >  22
FILE  .BAD     1 18-SEP-74   7723
```

Disks with many bad blocks can often be reused by reformatting them. First copy all desired files, since reformatting destroys all information contained an a volume.

## 4.3 PIP ERROR MESSAGES

The following error mesaages are output on the terminal when PIP is used incorrectlyr

| Errors | Meaninu |
|---|---|
| ?BAD BOOT? | A boot switch was specified on an illegal device. |
| ?BOOT COPY? | An error occurred during an attempt to write bootstrap with /U switch. |
| ?CHK SUM? | A checksum error occurred in a formatted binary transfer. |
| ?COR OVR? | Memory overflow--too many devices and/or file specifications (usually *.* operations) and no room for buffers. |
| ?DEV FUL? | No room on device for file. |
| ?ER RD DIR? | Unrecoverable error reading directory. Check volume for off-line or write-locked condition and try the Operation again. |
| ?ER WR DIR? | Unrecoverable error writing directory. Try again. |
| ?EXT NEG? | A /T command attempted to make file smaller. |
| ?FG PRESENT? | An attempt was made to use /O or /S while a foreground job was still in memory. Unload it if it is no longer desired. |
| ?FIL NOT FND? | File not found during a delete, copy, or re-name Operation, or no input files with the expected name or extension were found during a *.* expansion. |

| | |
|---|---|
| ?ILL CMD? | The command specified was not syntactically correct; a device name is missing which should be specified, a switch argument is too large, a filename is specified where one is inappropriate, or a nonfile-structured device is specified for a file-structured Operation. |
| ?ILL DEV? | Illegal or nonexistent device. |
| ?ILL DIR? | The device did not contain a properly initialized directory structure (EOT file on magtape and cassette; empty file directory on other devices). Use /Z. |
| ?ILL REN? | Illegal rename Operation. Usually caused by different device names on the input and output sides of the command string. |
| ?ILL SWT? | Illegal switch or switch combination. |
| ?IN ER? | Unrecoverable error reading file. Try again (this error is ignored during /G Operation). |
| ?OUT ER? | Unrecoverable error writing file. Perhaps a hardware or checksum error; try recopying file. Also may be caused by an attempt to compress a larger device to a smaller one or by not enough room when creating a file. The system takes the largest space available and divides it in half before attempting to insert the file. Try the [n] construction or /X switch. |
| ?OUT FIL? | Illegal output file specification or missing output file. |
| ?ROOM? | Insufficient space following file specified with a /T switch. |

The following warning messages are output by PIP:

**CTn:** PUSH REWIND OR MOUNT NEW VOLUME

A new cassette must be mounted on drive n to allow continuation of an I/O Operation. The Operation is continued automatically as soon as the new cassette is mounted.

| | |
|---|---|
| ?NO .SYS/.BAD ACTION? | The /Y switch was not included with a command specified on a .SYS or .BAD file. The command is executed for all but the .SYS and .BAD files. A *.* transfer is most likely to cause this message. |
| ?REBOOT? | .SYS files have been transferred, renamed, compressed or deleted from the system device. It may be necessary to reboot the system. |

NOTE

The message is typed immediately after execution of the relevant command has begun, but the actual reboot Operation must not be performed until PIP returns with the prompting asterisk for the next command. If the system is halted and rebooted before the prompting asterisk returns, disk information may be lost.

If any of the .SYS files in use by the current system (MONITR.SYS and handler files) have been physically moved on the system device, it is necessary to reboot the system immediately. ff not, this message can be ignored. If the cause of the message was a /S Operation, the system need be rebooted only if there was an empty space before any of the .SYS files or if thc /N:n switch wae used to increase the number of directory Segments. The need to reboot can be permanently avoided by placing all .SYS files at the beginning of the system device, then avoiding their involvements in PIP operations by not using the /Y switch.

dev:/Z ARE YOU SURE?

Confirmation must be given by the user before a device can be zeroed.

# CHAPTER 5

# MACRO ASSEMBLER

MACRO is a 2-pass macro assembler requiring an RT-11 system configuration (or background partition) of 12K or more. Macros are instructions in a source or command language which are equivalent to a spccified sequonce of machine instructions or commands. Users with minimum memory configurations must use ASEMBL and EXPAND and should read this chapter and Chapters 10 and 11 before assembling any programs. (The macro features not supported by ASEMBL are indicated in this chapter; many of the features not available in ASEMBL are supported by EXPAND.)

Some notable features of MACRO are:

1. Program control of assembly functions
2. Device and file name specifications for input and output files
3. Error listing on command output device
4. Alphabetized, formatted symbol table listing
5. Relocatable object modules
6. Global symbols declaration for linking among object modules
7. Conditional assembly directives
8. Program sectioning dirsctives
9. User defined macros
10. Comprehensive set of system macros
11. Extensive listing control, including cross reference listing

Operating instructions for the MACRO assembler appear in Section 5.7.

## 5.1 SOURCE PROGRAM FORMAT

A source program is composed of a sequence of source lines; each source line contains a single assembly language Statement followed by a Statement terminator. A terminator may be either a line feed Character (which increments the line count by 1) or a form feed Character (which resets the line count and increments the page count by 1).

NOTE

EDIT automatically appends a line feed to every carriage return encountered in a source program. For listing format, MACRO automatically inserts a carriage return before any line feed or form feed not already prcceded by one.

An assembly language line can contain up to 132(decimal) characters (exclusive of the Statement terminator). Beyond this limit, excess characters are ignored and generate an error flag.

### S.l.1 Statement Format

A Statement can contain up to four fields which are identified by order of appearance and by specified terminating characters. The general format of a MACRO assembly language Statement is:

```
label:    Operator operand(s) ;comments
```

The label and comment fields are optional. The Operator and Operand fields are interdependent; either may be omitted depending upon the contents of the other.

The assembler interprets and processes these Statements one by one, generating one or more binary instructions or data words or performing an assembly process. A Statement contains one of these fields and may contain all four types. Blank lines are legal.

Some Statements have one Operand, for example:

```
CLR     R0
```

while others have two:

```
MOV     #344,RZ
```

An assembly lanyuage Statement must be complete on one source line. No continuation lines are allowed. (If a continuation is attempted with a line feed, the assembler interprets this as the Statement terminator.)

MACRO source Statements may be fonnatted with EDIT so that use of the TAB Character causes the Statement fields to be aligned. For example:

```
Label       Operator    Operand          Comment
Field       Field       Field            Field

CHECK:      BIT         #1,R0            ;IS NUMBER ODD?
            BEQ         EVEN             ;NO,IT'S EVEN
            MOV         #-1,ODDFLG       ;ELSE SET FLAG
EVEN:       RTS         PC               ;RETURN
```

5.1.1.1 Label Field - A label is a user-defined symbol that is unique within the first six characters and is assigned the value of the current location counter and entered into the user-defined symbol table. The value of the label may be either absolute (fixed in memory indopendcntly of the position of the program) or relocatable (not fixed in memory), depending on whether the location counter value (see Section 5.2.6) is currently absolute or relocatable.

A label is a symbolic means of referring to a specific location within a program. If present, a label always occurs first in a Statement and must be terminated by a colon. For example, if the current location is absolute 100(octal), the Statement:

```
ABCD:   MOV     A,B
```

assigns the value 100(octal) to the label ABCD. Subsequont 'reference to ABCD references location 100(octal). In this example if the location counter was declared relocatable within the section, the final value of ABCD would be 100(octal) plus a value assigned by LINK when it relocates the code, called the relocation constant. (The final value of ABCD would therefore not be known until link-time. This is discussed later in this chapter and in Chapter 6.)

More than one label may appear within a single label field, in which case each label within the field is assigned the same value. For example, if the current location counter is 100(octal), the multiple labels in the Statement:

```
ABC:    ERREX:  MASK:   MOV     A,B
```

cause each of the three labels--ABC, ERREX, and MASK--to be equated to the value 100(octal).

A symbol used as a label may not be redefined within the ueer program. An attempt to redefine a label results in an error flag in the assembly listing.

5.1.1.2 Operator Field - An operator field follows the label field in a Statement and may contain a macro call, an instruction mnemonic, or an assembler directive. The Operator may be preceded by zero, one or more labels and may be followed by one or more operands and/or a comment. Leading and trailing spaces and tabs arc ignored.

When the Operator is a macro call, the assembler inserts the appropriate code to expand the macro. When the Operator is an instruction mnemonic, it specifies thc instruction to be generated and the action to be performed on any operand(s) which follow. When the Operator is an assembler directive, it specifies a certain function or action to be performed during assembly.

An Operator is legally terminated by a space, tab, or any non-alphanumeric Character (symbol component).

Consider the foiiowing examples:

```
MOV A,B   (space terminates the Operator MOV)
MOV@A,B   (@ terminatee the operator MOV)
```

When the Statement line does not contain an Operand or comment, the operatot is terminated by a carriage return followed by a line feed or form feed Character.

A blank Operator field is interpreted as a .WORD assembler directive (See Section 5.5.3.2).

5.1.1.3 Operand Field - An Operand is that part of a Statement which is manipulated by the Operator. Operands may be axptessions, numbers, or symbolic or macro arguments (within the context of the Operation). When multiple operands appear within a Statement, each is separated from the next by one of the following characters: comma, tab, space, or paired angle brackets around one or more operands (see Section 5.2.1.1). Multiple delimiters separating operands are not legal (with the exception of spaces and tabs--any combination of spaces and/or tabs represents a single delimiter). An Operand may be preceded by an Operator, a iabel or another Operand and followed by a comment.

The Operand field is terminated by a semicolon when followed by a comment, or by a Statement terminator when the Operand completes the Statement. For example:

```
LABEL; MOV A,B   ;COMMENT
```

The space between MOV and A tenninates the Operator field and begins the Operand field; a comma separates the operands A and B; a semicolon terminates the operand field and begins the comment field.

5.1.1.4 Comment Field - The comment field is optional and may contain any ASCII characters except null, rubout, carriage return, line feed, vertical tab or form feed. All other characters, even special characters with defined usage, are ignored by the assembler when appearing in the comment field.

The comment field may be preceded by one, any, none or all of the other three field types. Comments must begin with the semicolon Character and end with a Statement terminator.

Comments do not affect assembly processing or program execution, but are useful in source listings for later analysis, debugging, or documentation purposes.

### S.l.2 Format Control

Horizontal or line formatting of the source program is controlled by the space and tab characters. These characters have no effect on the assembly process unless they are embedded within a symbol, number, or ASCII text; or unless they are used as the Operator field terminator. Thus, these characters can be used to provide an orderly source program. A Statement can be written:

```
LABEL:MOV(SP)+,TAG;POP VALUE OFF STACK
```

or, using formatting characters, it can be writtent

```
LABEL: MOV     (SP)+,TAG        ;POP VALUE OFF STACK
```

which is easier to read in the context of a source program listing.

Vertical formatting, i.e., page size, is controlled by the form feed Character. A page of n lines is created by inserting a form feed (CTRL FORM) after the nth line. (See also Section 5.5.1.6 for a description of page formatting with respect to macros and Section 5.5.1.2 for a description of assembly listing output.)

## 5.2 SYMBOLS AND EXPRESSIONS

This section describes the various components of legal MACRO expressions: the assembler Character set, symbol construction, numbers, Operators, terms and expressions.

### 5.2.1 Character Set

The following characters are legal in MACRO source programs:

1. The letters A through z. Both upper- and lower-case letters are acceptable, although, upon input, lower-caae letters are converted to upper-case letters. Lower-case letters can only be output by sending their ASCII values to the output device. This conversion is not true for .ASCII, .ASCIZ, ' (single quote) or " (double quote) statements if .ENABL LC is in effect.

2. The digits 0 through 9.

3. The characters . (period or dot) and $ (dollar sign) which are reserved for use in system program symbols (with the exception of local Symbols; see Söction 5.2.5).

4. The following special characters:

| Character | Designation | Function |
|---|---|---|
| carriage return | | formatting Character |
| line feed | | |
| form feed | | source Statement terminators |
| vertical tab | | |
| : | colon | label terminator |
| = | equal sign | direct assignment indicator |
| % | percent sign | register term indicator |
| tab | | item or field terminator |
| space | | item or field terminator |
| # | number sign | immediate expression indicator |
| @ | at sign | deferred addressing indicator |
| ( | left parenthesis | initial register indicator |
| ) | right parenthesis | terminal register indicator |
| , | comma | Operand field separator |
| ; | semicolon | comment field indicator |
| < | left angle bracket | initial argument or expression indicator |
| > | right angle bracket | terminal argument or exprcssion indicator |
| + | plus sign | arithmetic addition Operator or auto incrament indicator |
| | minus sign | arithmetic subtraction Operator or auto decrement indicator |
| * | asterisk | arithmetic multiplication Operator |
| / | slash | arithmetic division Operator |
| & | ampersand | logical ARD Operator |
| ! | exclamation | logical inclusive OR Operator |
| " | double quote | double ASCII Character indicator |
| ' | single quote | single ASCII Character indicator |
| ↑ | uparrow | universal unary Operator, argument indicator |
| \ | backslash | macro numeric argument indicator (not available in ASEMBL) |

5.2.1.1 Separating and Delimiting Characters - Reference is made in the remainder of the chapter to legal separating characters and macro argument delimiters. These terms are defined in Table 5-l and following.

Table 5-l
Legal Soparating Characters

| Character | Definition . | Usage |
|---|---|---|
| space | one or more spaces | A space is a legal separator |

# CHAPTER 6

# LINKER

## 6.1 IETRODUCTION

The RT-11 Linker converts object modules produced by either one of the RT-11 assemblers or FORTRAN IV into a format suitable for loading and execution. This allows the user to separately assemble a main program and each of its subroutines without assigning an absolute load address at assembly time. The object modules of the main program and subroutines are proceesed by the Linker tot

1. Relocate each object module and assign absolute addresses
2. Link the modules by correlating global symbols defined in one module and referenced in another module
3. Create the initial control block for the linked program
4. Create an overlay structure if specified and include the necessary run-time overlay handlers and tables
5. Search user specified libraries to locate unresolved globals
6. Optionally produce a load map showing the layout of the load module

The RT-11 Linker requires two or three passes over the input modules. During the first pass it Constructs the global symbol table, including all control section names and global symbols in the Input modules. ff library files are to be linked with input modules, an intermediate pass is needed to force the modules resolved from the library file into the root segment (that part of the program which 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 (.LDA for use with the Absolute Loader, save image (.SAV) for a Single-job system or for the background job of an F/B System, and relocatable (.REL) format for the foreground iob of an F/B System).

The Linker runs in a minimal RT-11 system of BK; any additional memory is used to facilitate efficient linking and to extend the symbol table. Input is accepted from any random-access device on the system; there must be at least one random-access device (disk or DECtape) for save image or relocatable format output.

## 6.2 CALLING AND USING THE LINKER

To call the Linker, type the command:

```
R LINK
```

and the RETURN key in response to the Keyboard monitor's dot. The Linker prints an asterisk and awaits a command string.

Type CTRL C to halt the Linker at any time and return control to the monitor. To restart the Linker, type R LINK or the REENTER command in response to the monitor's dot. The Linker outputs an extra line feed Character when it ie restarted with REENTER or after an error in the first command line. When the Linker is finished linking, control returns to the CSI automatically. An extra line feed Character precedes the asterisk printed by the CSI.

### 6.2.1 Command String

The first command string entered in response to the Linker's asterisk has the following formt:

```
*dev:binout,dev:mapout=dev:objl,dev:obj2,.../sl/s2/s3
```

where :

| | |
|---|---|
| devr | is a random-access device for all files except dev:mapout, which can be any legal output device. If dev: is not specified, DK is assumed. If the output is to be LDA format (that is, the /L switch was used), the output file need not be on a random-access device. |
| binout | is the name to be assigned to the Linker'8 save image, LDA formt, or REL format output file. This file is optional; if not specified, no binary output is produced. (Save image is the assumed output format unless the /L or /R switches are used.) |
| mapout | is the optional load map file. |
| objl,... | are files of one or more object modules to be input to the *Linker* (these may be library files). |
| /sl/s2/s3 | are switches as explained in Table 6-1 and Section 6.8. |

If the /C switch is given, subsequent command lines may be entered ast

```
*objm,objn,.../sl/s2
```

The /C switch is necessary only if the command string will not fit on one line or if the overlay structure is used. If an error occurs in a continued command line (e.g., ?FILE NOT FND?), only the line in error need be retyped.